UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Doble Grado en Informática y Matemáticas

TRABAJO FIN DE GRADO

DESARROLLO DE UN SIMULADOR DE SISTEMAS DE MEMBRANAS

Daniel Sánchez-Carnerero Polo
Tutor: Jack Mario Mingo Postiglioni
Ponente: Gonzalo Martínez Muñoz

JULIO2016

DESARROLLO DE UN SIMULADOR DE SISTEMAS DE MEMBRANAS

AUTOR: Daniel Sánchez-Carnerero Polo
TUTOR: Jack Mario Mingo Postiglioni
PONENTE: Gonzalo Martínez Muñoz

Escuela Politécnica Superior
Universidad Autónoma de Madrid
Julio de 2016

Resumen

Este Trabajo Fin de Grado está basado en la simulación de un tipo concreto de P System, los denominados Spiking Neural P Systems (SN P Systems). Este tipo de sistemas de computación con membranas constan de un conjunto de neuronas que están conectadas uni o bidireccionalmente entre ellas y que contienen un conjunto de símbolos iguales, denominados *spikes*, que permiten la evaluación de una serie de reglas. Gracias a estas reglas, si se cumple cierta condición sobre los spikes que contiene la neurona en el ciclo de ejecución correspondiente, de una forma determinista o no, se elegirá una de las posibles reglas a evaluar y será ejecutada. La ejecución de una regla implicará el consumo de un número de spikes y el envío de spikes a las neuronas con las que presenta conexión la neurona que contiene la regla. De esta forma, a través del flujo síncrono de spikes a lo largo de todo el sistema, es posible interpretar la salida de lo que sería la ejecución completa de la simulación.

Para llevar a cabo la mencionada simulación se ha desarrollado una herramienta utilizando el lenguaje R, haciendo uso de la librería gráfica de Shiny. Con esta plataforma es posible cargar desde un fichero el diseño de un Spiking Neural P System, con el fin de ver gráficamente su estado, configurar los parámetros de la computación y proceder a su simulación. Ésta consistirá en representar el estado de todo el sistema para cada ciclo de reloj que se ejecute, pudiendo llevarse a cabo de ciclo en ciclo o por bloques de un número concreto de ciclos.

Por lo tanto, con la realización de este trabajo se ha desarrollado una herramienta que permite la simulación de Spiking Neural P Systems, bien para un número concreto de pasos, o bien para comprobar su funcionamiento paso a paso. La herramienta ofrece la posibilidad de usar sistemas extendidos, es decir, que emplean *spikes* con un tiempo de vida máximo, o que eliminan el contenido de toda la neurona cada vez que se ejecuta una regla.

En cuanto a la aplicación de estos sistemas con el simulador, se ha podido comprobar su uso en la simulación de puertas y circuitos lógicos, y en algoritmos de ordenación. Al margen de las aplicaciones de los Spiking Neural P System, el simulador permitirá la comprobación real de ciertos resultados teóricos que pueden ser empleados en diferentes líneas de investigación.

Palabras clave

P Systems, Spiking Neural P Systems, Sistemas de Reglas, Computación Natural y Computación con Membranas.

Abstract

This bachelor thesis is based on the simulation of a concrete type of P-System that is called Spiking Neural P-Systems (SN P Systems). That type of computing system of membranes consists of a set of neurons that are connected uni- or bidirectionally between each other and contain a set of elements formed by one pre-stablished symbol, called spikes, that permit the evaluation of rules. Those rules have a specific condition about the number of the spikes that they contain in a specific cycle. If the condition of more than one rule is met, the system will choose one of them in a non-determined way, one of them will be chosen and executed. The computation of a rule will consume the number of spikes indicated by the rule and it will send spikes to the neurons with whom the cell shares a connection. With this format, by means of a synchronized flow of spikes along the system, it is possible to interpret the output of what would be the complete execution of the simulation.

To process this simulation, a specific application using the programming language "R" has been developed, making use of the graphic library of "Shiny". By means of this platform it is possible to upload the design of a Spiking Neural System P from a file, with the goal of visualizing graphically its state, configurate the parameters of for the computation and proceeding with its simulation. The simulation will consist of the representation of the state of the whole system for every cycle, being able to compute step-by-step or by blocks of a concrete numbers of cycles.

Therefore, through the realization of this bachelor thesis, a specific application that allows the simulation of Spiking Neural P Systems has been elaborated, which enables the user to follow through each step of the process. It offers the possibility of working with extended systems, which means that it employs spikes with a maximum lifetime or that they eliminate the content of the cell when they spike.

Concerning the application of those systems, it has been possible to review their use in the simulation of gates and logic circuits and in algorithms of sorting. Apart from the application of SN P Systems in concrete areas, the simulator will be able to offer the user the possibility to verify certain theoretical results that can be applied to different fields of investigation (eg. Skeletonization of images, Neural networks, etc.)

Keywords

P Systems, Spiking Neural P Systems, Natural Computing, Membrane Computing, Systems of Rules

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1 Introducción

A lo largo de este trabajo vamos a desarrollar una herramienta que permita llevar a cabo la creación y simulación de ciertos sistemas de computación natural, en concreto los denominados Spiking Neural P Systems. Podemos entender este concepto como la capacidad que tiene la naturaleza para computar o bien para inspirar computación. Si lo vemos desde el primer punto de vista, ciertas líneas de investigación llevan a cabo estudios para que sean los sistemas vivos los que directamente realizan ciertas computaciones y analizan mecanismos reales con los que la naturaleza procesa información. Desde la segunda perspectiva, se definen dispositivos de computación abstracta que imitan los procesos, mecanismos y formas en que las diversas leyes de la naturaleza producen modificaciones en determinados sistemas (desde hábitats hasta conjuntos de moléculas, pasando por organismos vivos) que pueden ser interpretados como procesos de cálculo sobre sus elementos. Así como un hábitat en el que varias especies de seres vivos conviven, sufre transformaciones con el paso de las generaciones y en donde la interacción entre las especies puede provocar cambios en los elementos que la forman (cambios que pueden afectar desde la distribución de dichas especies en el hábitat hasta la morfología propia de cada especie). Todo ello es lo que entendemos como evolución de las especies; un conjunto de moléculas en un entorno con determinadas características (de temperatura, salinidad, etc.) puede evolucionar hacia estados estructuralmente más complejos modificando las características de las mismas; es decir, propiciando reacciones químicas entre ellas que pueden llegar a producir elementos funcionalmente más complejos, como son las moléculas de ácido desoxirribonucléico (ADN).

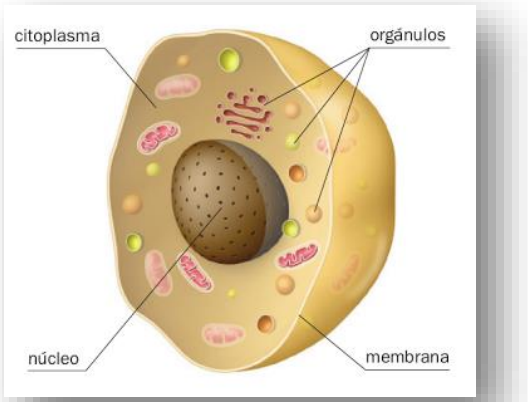


Figura 1 Imagen de una célula

Una de las primeras características que llama la atención en cuanto a la estructura interna de una célula (Figura 1), es el hecho de que las distintas partes que la componen se encuentran delimitadas por varios tipos de *membranas* (en su sentido más amplio), desde la propia membrana que separa el exterior del interior de la misma, hasta las distintas membranas que delimitan los componentes internos. Además, con respecto a la función en la naturaleza de estas membranas, interesa el hecho de que no generan compartimentos estancos sino que permiten el paso (flujo) de ciertos compuestos químicos, algunas veces de forma selectiva e incluso en una sola de las direcciones.

La *Computación Celular con Membranas* ha sido hasta ahora el último modelo de Computación Natural. Fue introducido por Gh. Paun[1] en octubre de 1998 como un modelo de tipo distribuido y paralelo, y está inspirado en el funcionamiento de la célula como organismo vivo capaz de procesar y generar información. A este tipo de sistemas que utilizan la computación celular con membranas se les denomina actualmente P Systems y dentro de ellos, una clase particular centrada en la modelización de neuronas es la denominada Spiking Neural P Systems, que constituye el objeto de este trabajo. Un estudio más detallado acerca de este tipo de sistemas y sus diferentes tipos de implementaciones se lleva a cabo en el capítulo 2.

1.1 Motivación

Los P Systems presentan tres partes fundamentales: las *células* que lo forman, las *reglas* contenidas en cada célula que emiten los elementos (químicos) que se transmiten entre ellas, y por tanto, sus *conexiones*.

Estos sistemas están inspirados en la estructura y en el funcionamiento de células vivas, es natural considerarlos como herramienta de modelaje de sistemas biológicos, dentro del marco de los "Systems Biology[2]", siendo una alternativa o complemento a los enfoques más clásicos como las ecuaciones diferenciales ordinarias.

La mayoría de las variantes de P Systems son computacionalmente universales. Esto se extiende incluso a variantes que no usan prioridades en las reglas a ejecutar, normalmente un aspecto fundamental para los P Systems.

Por todo lo anterior, como modelos de computación, los P Systems ofrecen la atractiva posibilidad de resolver problemas NP-completos en un tiempo menor al exponencial. Algunas variantes son conocidas por ser capaces de resolver problemas SAT3 en tiempo lineal, y debido a que todos los problemas NP-completos son equivalentes, esta capacidad se aplica a muchos problemas. Como no hay un método directo de implementación de un P System, su funcionalidad es emulada y, por tanto, resolver problemas NP-completos en tiempo lineal solo está probado teóricamente. Sin embargo, se ha probado que cualquier P System determinista puede ser simulado en una Máquina de Turing en tiempo polinómico [1].

Como se ha comentado, los trabajos relacionados con los Spiking Neural P Systems son eminentemente teóricos, por lo que la idea de realizar una herramienta que permita el diseño y ejecución de este tipo de sistemas es interesante para poder trabajar con ellos de forma cercana y real. Actualmente, existen diferentes herramientas enfocadas al diseño de P Systems genéricos, pero hay muy pocas específicas para los Spiking Neural P Systems. Por estas razones se desea implementar una herramienta capaz de trabajar con los Spiking Neural P Systems de forma más directa e intuitiva, ofreciendo a los investigadores en particular, y a cualquier usuario interesado en estos sistemas en

[1] Corresponding Member of The Romanian Academy (1997) http://www.imar.ro/~gpaun/
[2] http://staffwww.dcs.shef.ac.uk/people/M.Gheorghe/PSimulatorWeb/P_Systems_applications.htm
[3] https://en.wikipedia.org/wiki/Boolean_satisfiability_problem

general, un simulador que permita experimentar y desarrollar de forma práctica sus ideas

Dos ejemplos claros del uso de los Spiking Neural P Systems se pueden encontrar en circuitos lógicos y en algoritmos de ordenación de números naturales [2].Para el primero de los casos, simulación de puertas y circuitos lógicos, se comienza con la construcción de las puertas más simples, como lo son OR, AND y NOT. Esto se puede hacer con un número muy pequeño de neuronas y permiten generar multitud de circuitos de la forma que a continuación brevemente se muestra en la Figura 2.

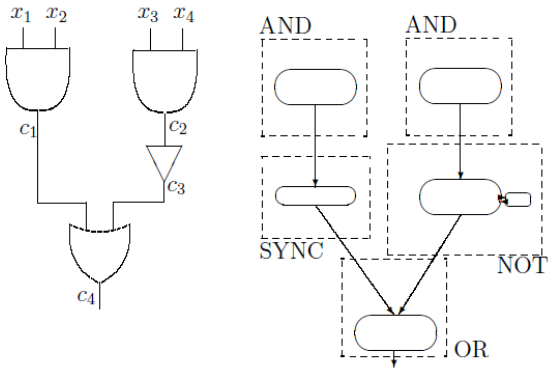


Figura 2 Circuito booleano y el Spiking System

1.2 Objetivos

Debido a la complejidad que presentan estos tipos de sistemas de Computación Natural, a lo largo del presente trabajo se va a realizar un estudio de los Spiking Neural P Systems con el fin de implementar un simulador capaz de realizar el diseño de uno de estos tipos de sistemas o recibirlo como entrada, para procesarlos y mostrar su salida en función de los parámetros dados. Se podrá llevar a cabo la simulación de los sistemas empleando reglas concretas, o permitiendo al sistema que elija entre las diferentes posibles de forma no determinista, por lo que conseguiremos diferentes salidas para entradas iguales de nuestro sistema. Toda la información se mostrará de forma gráfica en los diagramas con el fin de poder evaluar de forma más eficaz el funcionamiento para, si es necesario, realizar los cambios necesarios en la configuración inicial.

Todo lo anterior se podría resumir en un simulador que permita alcanzar los siguientes objetivos:

- Comprobar los resultados teóricos que puedan proponerse con el fin de permitir el avance en términos de investigación, especialmente en las líneas que centran su objetivo en que estos sistemas permitan desarrollar tareas como lo hacen ahora mismo las denominadas y conocidas redes neuronales.

- Permitir la creación y ejecución de Spiking Neural P Systems de forma real, mediante una sencilla interfaz que permita al usuario definir los parámetros de configuración relevantes y mostrar los resultados de manera gráfica, ofreciendo

además la posibilidad de modificar el diseño original y comprobar los nuevos resultados.

- Descubrir nuevas utilidades y destinos donde estos sistemas pueden resolver problemas de forma mucho más rápida de lo normal es un claro objetivo a marcarse. Podemos comprobar estas novedosas aplicaciones, por ejemplo, en la función de Skeletonization [3]. Dicha función consiste en reducir una forma en un conjunto de curvas, llamadas esqueletos, centradas en la forma original.

1.3 Organización de la memoria

Este trabajo se divide en ocho capítulos de los que, a continuación, se indica el contenido de cada uno de ellos.

El primer capítulo servirá de introducción y presentación del trabajo que se pretende llevar a cabo, ofreciéndose una visión general de la tarea a realizar y los objetivos marcados.

El segundo capítulo contiene los fundamentos teóricos de los sistemas que se estudian en este trabajo, dando una visión general de los P Systems, y centrando posteriormente los esfuerzos mediante la definición formal de los denominados Spiking Neural P Systems, las posibles extensiones de éstos, además de citar diferentes teoremas que derivan del uso de los SN P Systems.

En el tercer capítulo se aborda el estado del arte, donde se podrá comprobar las diferentes líneas de investigación que existen acerca del uso de los Spiking Neural P Systems, y algunas de las herramientas ya existentes actualmente.

Dentro del cuarto capítulo se dará una visión más detallada de las aplicaciones de los SN P Systems que son de mayor interés para este trabajo. Concretamente, se estudiarán las puertas y circuitos lógicos, y los algoritmos de ordenación de números.

En los capítulos quinto y sexto se presentará el Análisis y el Diseño del simulador de SN P Systems implementado.

En el séptimo capítulo se mostrarán las diferentes pruebas realizadas y resultados obtenidos con el fin de comprobar el funcionamiento del simulador.

En el octavo capítulo se presentan las conclusiones y el posible trabajo futuro.

2 Fundamentos teóricos

2.1 Conceptos generales. Introducción a los P Systems.

¿Qué es la Computación Celular con Membranas?
La Computación Celular con Membranas (de ahora en adelante, CCM) es un paradigma de computación inspirado en la estructura y el funcionamiento de las células de los organismos vivos. Su objetivo es observar la célula y diseñar máquinas o dispositivos que proporcionan herramientas novedosas para atacar, principalmente, problemas relevantes de la vida real, como construir modelos computacionales de procesos biológicos. Así se podría saber, por ejemplo, cómo se produce la comunicación inteligente en las bacterias.

¿De dónde viene la teoría de la CCM?
Surgió de la observación de la computación molecular en el ADN: después de un período inicial entusiástico, se vio que, en las aplicaciones prácticas, el ADN se comportaba 'mal' en el laboratorio y 'mejor' en su estructura natural, la célula. Así que la idea fue observar directamente la célula como un conjunto y no sólo el genoma.

En una **célula** viva, cada membrana trabaja con compuestos químicos de acuerdo con unas reacciones químicas. En una **máquina** paralela, cada procesador trabaja con datos de acuerdo con un programa específico. La Tabla 1 muestra la equivalencia de los conceptos entre el dispositivo natural y el artificial.

Célula	Máquina
Membranas	Procesadores
Compuestos químicos	Datos
Reacciones químicas	Instrucciones

Tabla 1 Tabla comparativa Célula-Máquina

¿Qué aplicaciones puede tener la CCM?

A nivel práctico, destacan sobre todo las aplicaciones de la CCM en biología, biomedicina y ecología. Es importante porque los biólogos necesitan herramientas y técnicas adecuadas para tratar sus datos empíricos. De hecho, grupos de investigación de Sevilla, Milán y Verona (ambas en Italia), ya hacen aplicaciones de este tipo.

Dos ejemplos de las aplicaciones hechas por el grupo de investigación de Sevilla, en colaboración con un equipo de Lérida, son la gestión de los ecosistemas del quebrantahuesos (*Gypaetus barbatus*) en el Pirineo Catalán y del mejillón cebra (*Dreissena polymorpha*) en el pantano de Ribarroja en la zona del Ebro. Cabe destacar que aparte de estas aplicaciones prácticas, existen muchas aplicaciones teóricas.

Para llevar a cabo la computación celular, ¿qué elementos son esenciales?
Los P Systems, nombre con el que se conoce a los dispositivos computacionales (o máquinas) del paradigma de la computación celular con membranas, son herramientas teóricas con un cierto aspecto físico (equivalente a una sintaxis) y que se comportan de una determinada forma (equivalente a una semántica). Esas máquinas celulares teóricas se simulan mediante aplicaciones software diseñadas ad hoc para utilizarlas en aplicaciones prácticas como la génesis de tumores.

2.2 P Systems

Los P Systems se definen como una serie de membranas que contienen químicos (en cantidades finitas), catalizadores y reglas que determinan los posibles caminos que pueden tomar los químicos cuando reaccionan. Las reglas también pueden causar que los químicos salgan al exterior de la membrana o que ésta se disuelva.

Aunque existen muchas variedades de P Systems, la mayoría comparte los mismos componentes básicos:

- El **entorno**: se define como los alrededores del P System. En el estado inicial, alberga solamente al contendedor de la membrana más externa, pero nunca contiene reglas. Los objetos que se encuentran en el entorno al final de la computación constituyen el resultado o porte de él. Alguna de las variantes de estos sistemas permite la división de membranas, la posesión de carga o tienen permeabilidad variable a través de cambios en el espesor de la membrana.

- Las **membranas**: son las estructuras principales dentro de un P System. Una membrana es una unidad discreta que puede contener un conjunto de objetos (símbolos / catalizadores), un conjunto de reglas, y un conjunto de otras membranas. La membrana más externa situada en la capa del entorno, se le denomina como “skin membrane”. Tal como indica su nombre, las membranas son permeables y los símbolos resultantes de una regla pueden cruzarlas. Una membrana, no la externa, también puede ser disuelta, en cuyo caso, su contenido se transfiere hacia la membrana que la contenía.

- Los **símbolos**: representan los productos químicos que pueden reaccionar con otros productos químicos para formar algún producto. En un P System cada tipo de símbolo está típicamente representado por una letra diferente. Por lo tanto, el contenido del símbolo de una membrana está representado por una cadena de letras. Debido a la multiplicidad de símbolos en una región, se utilizan comúnmente multiconjuntos para tal fin.

  Existen símbolos de casos especiales, como por ejemplo, δ, que se utiliza a menudo para indicar la disolución de una membrana, apareciendo siempre en la parte derecha de la regla. Una vez encontrado, provoca una reacción y se utiliza en el proceso de computación.

- Las **reglas**: representan una posible reacción química dentro de una membrana, haciendo que el sistema evolucione a un nuevo estado. Una regla tiene un conjunto de objetos de entrada (símbolos o catalizadores) que deben estar presentes con el fin de poder ejecutarse dicha regla. Si los objetos necesarios están presentes, los consume y produce un conjunto de objetos de salida una vez ejecutada. Una regla también puede ser especificada para que tenga prioridad sobre otras reglas, en cuyo caso, las reglas menos dominantes se aplicarán únicamente cuando no es posible aplicar una regla de mayor importancia. Por otro lado, se indica el retardo con el que la regla puede ser ejecutada efectivamente, es decir, que ésta quedará en estado "cerrado" durante los ciclos que se indiquen, no pudiendo recibir ni emitir nuevos spikes hasta que transcurra dicho retardo, momento en que podrá de nuevo enviar y recibir.

  Hay tres formas distintas en las que una regla puede manejar su salida. Normalmente los objetos de salida pasan a la misma membrana donde se encuentra la regla y las entradas, conocidas como "*here rule*". Sin embargo, hay dos modificadores que pueden ser especificados cuando las reglas se definen como "*in out*". El modificador "*in*" hace que el objeto de salida pase a la membrana hija, siendo elegida de manera aleatoria durante la computación. El modificador "*out*" causa que el objeto se dirija a la membrana padre o hijas de ésta, especificada en la configuración.

- Los **catalizadores**: son similares a sus homónimos en la química. Están representados y utilizados de la misma manera con símbolos, pero nunca se consumen durante una "reacción", y son simplemente un requisito más para que esta ocurra.

Por lo tanto, los ingredientes básicos de los P Systems son:

- Un **alfabeto**, cuyos elementos se denominan **objetos**.
- Una **estructura de membranas** (regiones)
- Un multiconjunto asociado a cada región
- Un conjunto de **reglas de evolución**
- Dos membranas distinguidas: una de **entrada** y otra de **salida**.

Por otro lado, se definen los conceptos de:

- **Configuración:** donde se especifican todos los parámetros del sistema, es decir, las membranas que lo componen y su disposición en el entorno, las reglas que contiene cada una de ellas, la entrada y salida del sistema, y los símbolos que se encuentran presente en cada membrana al comienzo de la simulación.

- **Computación:** es una sucesión finita o infinita de configuraciones. El momento de parada se podrá definir por un número concreto de ciclos de ejecución o debido a la estabilización del sistema, entendiendo esto como el punto en el que ya no es posible aplicar ninguna de las reglas del sistema, por lo que el estado nunca cambiará en ciclos posteriores.

- **Transición:** se denominará así al cambio de una configuración a otra cuando sea posible realizar este proceso, a través de la ejecución de las reglas posibles en dicho momento en el sistema.

Una vez se ha explicado el concepto de P System, y debido a la amplitud de este concepto y sus aplicaciones, a lo largo del presente trabajo se va acotar el problema a los ya mencionados Spiking Neural P Systems. En el siguiente apartado se definirán detalladamente estos sistemas, pero a modo de resumen, éstos serán un tipo de P System donde se dan las siguientes características:

- Las membranas que forman el sistema se encontrarán todas en el entorno, en un mismo nivel, es decir, no se permite que existan células dentro de otras células. En adelante, debido a este hecho, podremos referiremos a tales membranas como neuronas.

- El alfabeto que se va a emplear lo formará un único elemento que representaremos como "a".

2.3 Spiking Neural P Systems (SN P Systems)

Denominaremos Spiking Neural P Systems de grado $m \geq 1$ a un P System de la forma:

$$\Pi = (O, \sigma_1, \dots, \sigma_m, syn, i_o),$$

donde:

- $O = \{a\}$ es un alfabeto formado por un único elemento (al elemento a se le denomina *spike*);

- $\sigma_1, \dots, \sigma_m$ son neuronas de la forma:

  $$\sigma_i = (n_i, R_i), 1 \leq i \leq m,$$

  donde:

  - $n_i \geq 0$ es el número inicial de *spikes* en la neurona$i$;

  - R_i es un conjunto finito de reglas con las dos posibles formas siguientes:

    i. $E/a^r \rightarrow a; t$, donde E es una expresión regular,$r \geq 1$, and $t \geq 0$ Por ejemplo, si tenemos una regla de la forma $a^3/a \rightarrow a; 0$ nos indica que la regla sólo podrá ser ejecutada cuando la neurona contenga tres spikes, consumirá uno y no tiene ningún retardo a la hora del envío de su salida. A estas reglas se les denominan reglas de disparo.

    ii. $a^s \rightarrow \lambda$ para algún $s \geq 1$, con la restricción de que $a^s \notin L(E)$ para ninguna regla $E/a^r \rightarrow a; t$ del tipo (i) de R_i

    A estas reglas se les denomina reglas de olvido, puesto que si la regla es de la forma $a^3 \rightarrow \lambda$, esto indica que se consumirán tres spikes de la

neurona y no se emitirá ninguno. Este tipo de regla sólo se podrá usar cuando no pueda ejecutarse ninguna de las del tipo anterior.

- $syn \subseteq \{1,2,\dots,m\}x\{1,2,\dots,m\}$ con $(i,i) \notin syn$ para $1 \leq i \leq m$ (conexiones entre células)
- $i_0 \in \{1,2,\dots,m\}$ indica la neurona de salida.

A continuación, la Figura 3 muestra el esquema de un SN P System que contiene siete neuronas.

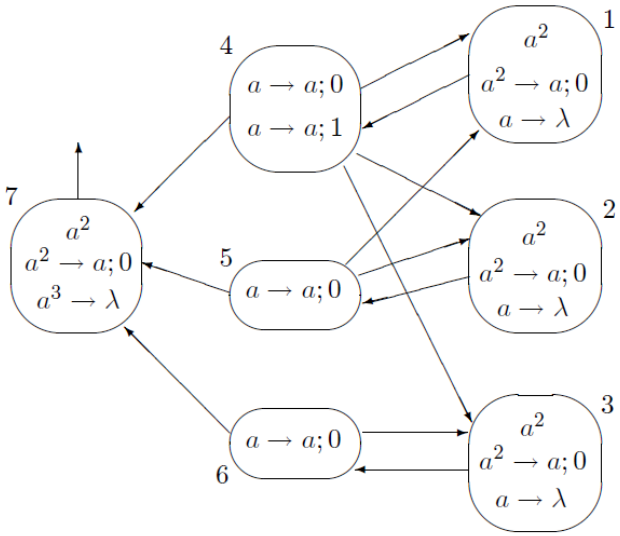


Figura 3 Esquema de un SN P System de ejemplo

Por otro lado, diremos que si C_1 y C_2 son dos configuraciones de Π, C_2 se obtiene de C_1 en **un paso de transición**$(C_1 \underset{\Pi}{\Rightarrow} C_2)$, si se puede pasar de C_1 a C_2 usando las reglas R_i que forman parte del SN P System.

2.4 Computaciones en SN P Systems

Una **computación**C es una sucesión (finita o infinita) de configuraciones $C_0 \underset{\Pi}{\Rightarrow} C_1 \dots \underset{\Pi}{\Rightarrow} C_r$, con $r \geq 0$, tal que:

- C_0 es la configuración inicial de Π
- Para cada $i \geq 0$, C_{i+1} se obtiene de C_i por un paso de transición.

Una computación, C, es de **parada** si $r \in \mathbb{N}$ y en la configuración C_r (denominada final) no hay reglas que se puedan aplicar.

2.5 Spiking Neural P System Extendidos (ESNP Systems)

Una vez que se ha indicado la caracterización de los Spiking Neural P Systems, veremos dos versiones extendidas de éstos que han sido propuestas por distintos autores [4].

- ESNP Systems con spikes con tiempo de vida limitado (*decaying spikes*)

  En este caso los spikes que contienen las neuronas poseen un tiempo de vida máximo, entendiendo éste como el máximo número de ciclos que pueden transcurrir sin ser consumido dicho spike antes de desaparecer de la neurona. Ahora un spikes "a" pasará a representarse de la forma (a, e) donde $e \geq 1$ indica dicho su tiempo de vida. En cada ciclo de computación este valor disminuye en una unidad hasta que, cuando $e = 0$, el correspondiente spike ya no figura en la neurona y no puede ser usado más adelante.

- ESNP Systems con Spiking total (*total spiking*)

  En este tipo de sistemas, el contenido completo de la neurona se borra cuando una regla del tipo (i) se ejecuta. En estos casos, se omite en la sintaxis de la regla que indica el número de spikes que se consumen en su ejecución.

Con estos tipos de SN P Systems extendidos, se puede pensar también en los denominados *ESNP Systems with thresholds*, donde los conjuntos regulares E son todo de la forma $\{n \in \mathbb{N} : n \geq h\}$, con h siendo denominada *threshold*. En este caso, las condiciones de ejecución de las reglas indican que la neurona ha de contener un número h de spikes concreto para que dicha regla pueda ser ejecutada.

2.6 Teoremas

Tras estudiar la estructura y la definición de los Spiking Neural P Systems, pasaremos a enunciar diferentes resultados teóricos que se consideran importantes. Una explicación más extensa y pruebas se pueden encontrar en [4].

2.6.1 Caracterización de los conjuntos finitos (NFIN)

Estudiaremos aquí la potencia de los SN P Systems con sólo una o dos neuronas:

Los SN P Systems con una neurona sólo pueden computar un conjunto finito de números, puesto que no es posible la retroalimentación, y cada computación dura como máximo tantos pasos como números de spikes hubiera inicialmente en la neurona. Por lo tanto, un primer resultado es que *cualquier conjunto finito puede ser computado por un sistema de una única neurona*. Tomamos un conjunto finito de números, $C = (n_1, n_2, \dots, n_k)$,y construimos el sistema $\Pi_C$con una neurona, conteniendo inicialmente dos spikes, con las siguientes reglas: $a^2/a \to a; 0$ y $a \to a; n_i$, para cada $i = 1,2,\dots,k$. El sistema emitirá un spike en el primer paso haciendo uso de la primera regla, a continuación, como aún queda un spike, vuelve a emitir un spike haciendo uso de una

de las reglas de la forma $a \to a; n_i$. Esto significa que el próximo spike será enviado en el ciclo $n_1 + 1$, por lo que $N_2(\Pi_C) = C$.

¿Qué ocurre con los sistemas formados por dos neuronas? Una de ellas debería ser la de salida y ésta puede emitir spikes sólo dos veces. Esto significa que esta neurona puede aumentar el número de spikes en el sistema como máximo en dos. La otra neurona no tiene conexiones consigo misma, por lo que como en el caso anterior, no puede usar más reglas que el número inicial de spikes presentes en ella, quizás esa cantidad más dos, debido a aquéllos que emita la neurona de salida. Todo esto significa que todas las computaciones están acotadas en longitud, por lo que el conjunto de números que pueden ser simulados por el sistema, de nuevo, es finito.

Denotaremos por $Spik_2P_m(rule_k, cons_p, forg_q)$ a la familia de todos los conjuntos que se puede obtener por un SN P System con un máximo de $m \geq 1$ neuronas, usando como máximo $k \geq 1$ reglas en cada neurona, con todas las reglas de la forma $E/a^r \to a; t$ teniendo $r \leq p$, y todas las reglas de olvido de la forma $a^s \to \lambda$ teniendo $s \leq q$. Cuando alguno de estos parámetros m, k, p, q no está acotado, se reemplazará con un asterisco *.

Se sintetizan estas observaciones en el siguiente teorema:

- $NFIN = Spik_2P_1(rule_*, cons_1, forg_0) = Spik_2P_2(rule_*, cons_*, forg_*)$.

Esto quiere decir que todos los conjuntos finitos pueden ser generados por SN P Systems de 1 neurona, con un número mayor que 1 de reglas que consumen menos de 1 spike en su ejecución y que no tienen reglas de olvido.

Una vez dado este resultado para el caso de sistemas de una o dos neuronas, generalicemos en el caso de que no existan restricciones en el número de neuronas o en otros parámetros.

2.6.2 Completitud computacional

Definiremos a la familia de los lenguajes recursivamente numerables como RE y a la familia de todos los conjuntos de números naturales Turing – computables como NRE. Con estas definiciones, veremos los siguientes resultados:

- *Cada conjunto del NRE puede ser computado por un SN P System Π teniendo sólo una neurona con reglas que pueden ser usadas de una forma no determinista.*

- $Spik_2P_m(rule_k, cons_p, forg_q) \subseteq Spik_2P_{m'}(rule_{m'}, cons_{p'}, forg_{q'}) \subseteq Spik_2P_*(rule_*, cons_*, forg_*) \subseteq NRE, para\ todo\ m' \geq m \geq 1, k' \geq k \geq 1,\ p' \geq p \geq 1, q' \geq q \geq 0$.

- $Spike_2P_*(rule_k, cons_p, forg_q) = NRE, para\ todo\ k \geq 2, p \geq 3, q \geq 3$.

2.6.3 Una caracterización de los conjuntos semilineales de números

Intentemos ser más "realistas", no permitiendo que las neuronas mantengan un número elevado de spikes al mismo tiempo. Esto recuerda a la función sigmoidal relacionando la entrada con la neurona excitada, donde después de un umbral dado, los spikes adicionales no importan, y simplemente son ignorados.

Un resultado relevante es que los SN P Systems que tienen acotado el número de spikes que pueden mantener durante una computación, generan conjuntos semilineales de números naturales[5].

Denotemos con $Spik_2P_m(rule_k, cons_p, forg_q, bound_s)$ a la familia de conjuntos de números $N_2(\Pi)$ computados por SN P Systems Π con m celulas como máximo, usando k reglas a lo sumo, consumiendo como mucho p spikes y olvidando menos de q spikes en cada regla, y con a lo sumo s spikes presentes en una neurona en cualquier momento en cualquier neurona (si una computación alcanza una configuración donde una neurona acumula más de s spikes, entonces se aborta la computación). Reemplazaremos con * los parametros que no están acotados ($bound_*$ significará que consideramos sólo SN P Systems con un número acotado de spikes las neuronas, pero esta cota no está especificada; cuando $bound_\alpha$no está presente en esta notación, ésto significará que el número de spikes en las neuronas pueden crecer arbitrariamente sin límite).

Esto da como consecuencia los siguientes teoremas:

- $SLIN_1 = Spik_2P_*(rule_k, cons_p, forg_q, bound_s), \forall k, q, p, s \geq 3.$
- $Spik_2P_*(rule_k, cons_p, forg_q, bound_s) \subseteq SLIN_1$
- $Cada\ singleton\ \{n\}, n \geq 1, es\ un\ Spik_2P_1(rule_2, cons_1, forg_0, bound_2)$
- *Toda progresión aritmética de la forma* $\{n_i | i \geq 1\}, n \geq 3$, *es un* $Spik_2P_{n+2}\ (rule_3, cons_3, forg_2, bound_3)$.
- $Si\ Q \in Spik_2P_m(rule_k, cons_q, forg_p, bound_s)\ y\ r \geq 1,\ entonces,$ $\{x + r \mid x \in Q\} \in Spik_2P_{m+1}(rule_k, cons_p, forg_p, bound_s),\ para\ todo\ m \geq 1\ ,$ $k \geq 2, p \geq 3, q \geq 0, s \geq 3.$
- $Si\ Q_1, Q_2 \in Spik_2P_m(rule_k, cons_p, forg_p, boud_s),\ \ para\ alg\acute{u}n\ m \geq 1,\ k \geq 2,$ $p \geq 2, q \geq 1, s \geq 2, entonces\ Q_1 \cup Q_2 \in$ $Spik_2P_{2m+6}(rule_k, cons_p, forg_q, bound_s)$
- $Existen\ conjuntos\ Q_1, Q_2\ de\ la\ familia\ Spik_2P_3(rule_3, cons_3, forg_2, bound_3)$ $tal\ que\ Q_1Q_2 = \{nm | n \in Q_1, m \in Q_2\}$ *que no están en* $Spik_2P_*(rule_*, cons_*, forg_*, bound_*)$.

2.7 Systems trabajando en un modo aceptación

Un SN P System también puede ser usado en un modo aceptación. Consideramos la siguiente manera de introducir el número a ser aceptado: la neurona especial i_0 es usada ahora como una neurona de entrada, pudiendo recibir spikes del entorno del sistema; asumimos que exactamente dos spikes entran en el sistema; el número n de pasos transcurridos entre los dos spikes será la salida del sistema; si, después de recibir los dos spikes, el sistema se detiene (no necesariamente en el momento de recibor el segundo spike), se acepta el número n. Denotemos con $N_{acc}(\Pi)$ al conjunto de números aceptados por el sistema Π, y con $Spik_{acc}P_m(rule_k, cons_p, forg_q)$ a la familia de conjuntos de esta forma correspondiente a la familia $Spik_2P_m(rule_k, cons_p, forg_q)$.
En el modo de aceptación podemos imponer la restricción de que en cada neurona sólo se puede aplicar a lo sumo una regla en cada unidad de tiempo, de ahí que el sistema se comporte de forma determinista. Cuando se consideren sólo SN P Systems deterministas, a la notación $Spik_{acc}P_m(rule_k, cons_p, forg_q)$ se le añadirá la letra D al comienzo.

Relacionando los resultados anteriores en este apartados, tenemos que:

- $DSpik_{acc}P_*(rule_k, cons_p, forg_q) = NRE\ para\ todo\ k \geq 2, p \geq 3, q \geq 2.$

2.8 Ejemplo de funcionamiento de los SN P Systems

A través del siguiente ejemplo, Figura 4, estudiaremos el funcionamiento de los SN P System, para cada ciclo de ejecución. Los resultados se pueden observar también en la Tabla 2.

En este ejemplo se simulará un SN P System que genera todos los números naturales pares. Para ello, comprobemos el contenido de cada neurona y las reglas que se aplican para los primeros pasos de la ejecución:

- $t = 0$

En el instante de tiempo inicial, las neuronas 1, 2, 3 y 7 contienen dos spikes.

- $t = 1$

Se ejecutarán las reglas de la forma $a^2 \rightarrow a; 0$ de las neuronas 1, 2, 3 y 7. Esto implica que se consumirá de todas ellas los dos spikes que contienen y no presentarán ningún spike más. Debido a la ejecución de las reglas, las neuronas 4, 5 y 6 recibirán un spike provenientes de las células 1, 2 y 3, respectivamente. Teniendo en cuenta que la neurona 7 es la de salida, por lo que comenzaremos a contabilizar el número de ciclos que transcurren hasta que vuelva a emitir *spikes*. Dicho número será un resultado parcial de la simulación.

- $t = 2$

Tengamos en cuenta que la neurona 4 presenta dos reglas con las mismas condiciones para poder ser evaluadas. Aunque el sistema de forma predeterminada elegirá cuál usar de forma aleatoria, en este caso, para seguir el camino de éxito buscado con la simulación, la generación de todos los números pares, elegiremos la regla $a \rightarrow a;0$. Debido a esto se ejecutarán las reglas de la forma $a \rightarrow a;0$ de las neuronas 4, 5 y 6, quedando sin *spikes* en este instante de tiempo.

Por la ejecución de las reglas del paso anterior, las neuronas 1, 2 y 3 presentarán ahora 2 spikes, y la neurona 7 presentará 3 spikes.

- $t = 3$

En este ciclo se ejecutarán las reglas de la forma $a^2 \rightarrow a;0$ de las neuronas 1, 2 y 3, quedando sin *spikes,* y la neurona 7 aplicará su regla $a^3 \rightarrow \lambda$, consumiendo sus tres *spikes*.Sólo las neuronas 4, 5 y 6 presentarán 1 *spike* en este ciclo.

- $t = 4$

En este instante vuelve a ocurrir lo mismo que en $t = 2$ puesto que la neurona 4 contiene 1 spike y puede aplicar cualquiera de las dos reglas que contiene. Por la misma razón expuesta anteriormente, se elegirá ahora la regla $a \rightarrow a;1$. Recordemos que esta regla se hará efectiva con un ciclo de retardo con respecto a las demás ejecutadas en este instante sin retardo. Se ejecutarán además las reglas $a \rightarrow a:0$ de las neuronas 5 y 6.

La neurona 1, 2 y 3 contendrán 1 spike cada una debido a la ejecución de las reglas del paso en $t = 3$, y la neurona 7, la de salida, contendrá 1 spike proveniente de la neurona 5 y otro de la neurona 6.

- $t = 5$

Las neuronas 1, 2 y 3 ejecutarán sus reglas de olvido ("*forgetting rules"*) de la forma $a \rightarrow \lambda$ por lo que consumirán sólo uno de sus dos spikes. La neurona 7 ejecutará su regla $a^2 \rightarrow a;0$ y emitirá un spike. Debido a lo indicado anteriormente, se tiene en cuenta el número de ciclos transcurridos y nos arroja la cifra de 4, por lo que se obtendrá el número 4 como salida. Las neuronas 1, 2 y 3 contendrán un spike, como hemos indicado, al igual que la neurona 7.

- $t = 6$

Se ejecutarán de nuevo las reglas de olvido de las neuronas 1, 2 y 3 y no contendrán ningún spike. No se puede ejecutar ninguna regla más y la computación se detiene. El número simulado es el 4 puesto que han transcurrido 4 ciclos de reloj entre las dos emisiones de spikes de la neurona de salida. La neurona 7 continúa con el spike que tenía ya en el paso anterior.

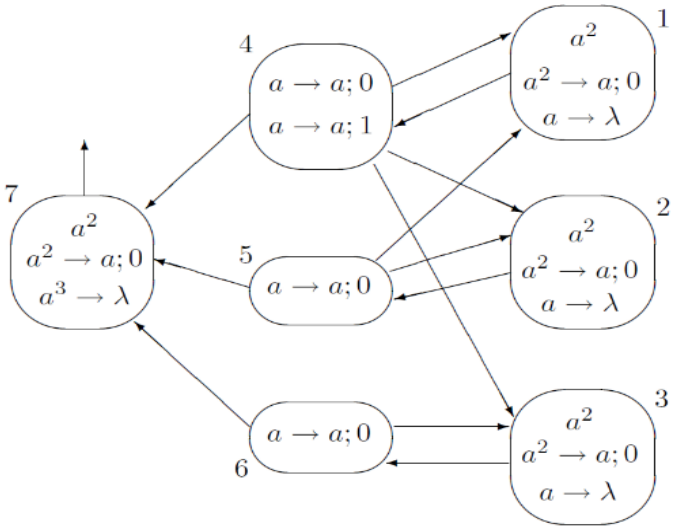


Figura 4 Un SN P System que genera todos los números naturales pares

Recogemos en la Tabla 2 lo explicado anteriormente para cada ciclo de ejecución.

Step / Neuron	0	1	2	3	4	5	6
1	aa	$a^2 \to a; 0$ —	— a_4a_5	$a^2 \to a; 0$ —	— a_5	$a \to \lambda$ a_4	$a \to \lambda$ —
2	aa	$a^2 \to a; 0$ —	— a_4a_5	$a^2 \to a; 0$ —	— a_5	$a \to \lambda$ a_4	$a \to \lambda$ —
3	aa	$a^2 \to a; 0$ —	— a_4a_6	$a^2 \to a; 0$ —	— a_6	$a \to \lambda$ a_4	$a \to \lambda$ —
4	—	— a_1	$a \to a; 0$ —	— a_1	$a \to a; 1$ —	— —	— —
5	—	— a_2	$a \to a; 0$ —	— a_2	$a \to a; 0$ —	— —	— —
6	—	— a_3	$a \to a; 0$ —	— a_3	$a \to a; 0$ —	— —	— —
7	aa	$a^2 \to a; 0$! —	— $a_4a_5a_6$	$a^3 \to \lambda$ —	— a_5a_6	$a^2 \to a; 0$! a_4	— a

Tabla 2 Resultado de la simulación del SN P System de la Figura 4

Vistos los primeros pasos de la simulación, la pregunta lógica es la siguiente: ¿cómo continuamos para simular todos los números naturales pares? Para lograr esta salida, la neurona 4 jugará un importante papel. Si se elige que use su regla $a \to a; 0$, y esto lo hacemos para $m \geq 0$ pasos, logramos que el sistema no emita ningún spike en la neurona de salida, por lo que podemos repetir estos pasos un número arbitrario de veces. Para que la neurona de salida vuelva a enviar un spike, debemos aplicar la regla $a \to a; 1$. Si esto ocurre en el paso t, sólo las neuronas 5 y 6 emiten spikes. Las neuronas 1, 2 y 3 reciben sólo un spike y es eliminado por sus reglas de olvido en el paso siguiente, $t + 1$. Por tanto, la neurona 7 recibe dos spikes, ejecuta su regla $a^2 \to a; 0$ y emite el

segundo spike al exterior. Esto ocurre en el momento $t + 1 = 2m + 2 + 1$, por lo tanto el número computado es el $2m + 2$, para algún $m \geq 0$. El spike de la neurona 4 cuyo envío se realizó indicando un retardo de 1 ciclo, llegará a las neuronas 1, 2, 3 y 7 en el paso $t + 1$, por lo que podrá ser usado en el siguiente, $t + 2$. En este instante, las neuronas 1,2 y 3 usan sus reglas de olvido y la computación termina.

3 Estado del arte en el desarrollo de simuladores para P Systems

3.1 P-Lingua

Entre los grupos de investigación interesados en las herramientas de desarrollo de aplicaciones para P Systems, el Grupo de Computación Natural de la Universidad de Sevilla ha desarrollado P-Lingua, un lenguaje de especificación de P Systems que interesa que se convierta en el lenguaje de programación estándar para este tipo de dispositivos. Una de sus principales características es proporcionar un lenguaje de programación lo más cercano posible a la notación formal utilizada en la literatura. El objeto es que los investigadores no tengan que hacer un esfuerzo suplementario para re-codificar utilizando una sintaxis nueva los P Systems que hayan diseñado. Este mismo grupo también ha desarrollado los módulos Java necesarios para simular todos los P Systems habitualmente presentes en la teoría. Pueden encontrarse más detalles en [6].

A pesar de que puede encontrarse la definición completa de la sintaxis en dicha referencia, mostraremos un fragmento a continuación comentando un ejemplo típico en esta materia que calcula cuadrados de números naturales de manera aleatoria.

```
@model<transition>
def main() {
        call n_cuadrados();
}
def n_cuadrados() {
        @mu = [[[]'3 []'4]'2]'1;
        @ms(3) = a,f;
        [a --> a,bp]'3;
        [a --> bp,@d]'3;
        [f --> f*2]'3;
        [bp --> b]'2;
        [b []'4 --> b [c]'4]'2;
        (1) [f*2 --> f ]'2;
        (2) [f --> a,d]'2;
 }
```

Figura 5 Ejemplo de definición de un P System

En la primera línea de la Figura 5se especifica el tipo de P System (en este caso de transición). Pueden definirse módulos que pueden ser llamados desde otros (línea3). En concreto **main** (línea 2) es el nombre usado para el módulo principal. La estructura se especifica mediante la palabra reservada **@mu**, con las membranas representadas mediante corchetes. Cada membrana es seguida por su identificador. Los contenidos de una membrana se especifican mediante la palabra reservada **@ms** que contiene entre paréntesis el identificador de la membrana descrita. Las reglas se representan a continuación. En el caso de que se necesite especificar prioridades en las reglas, se especifica el orden entre paréntesis antes de la propia regla.

3.2 GPUs para la implementación de versiones de P Systems

Por otro lado, algunos grupos de investigación han comenzado a considerar el uso de las llamadas GPUs para la implementación de versiones específicas de P System que se ejecuten de manera paralela. Hay que reconocer que mediante esta aproximación se pueden conseguir rendimientos temporales espectaculares tratando cada P System de manera adecuada. Todo ello con el fin de ofrecer a la comunidad un mecanismo más general para el acceso a recursos masivamente paralelos. Más información al respecto se puede encontrar en [7].

3.3 Simulador de Spiking Neural P Systems

Destacamos el Simulador de Spiking Neural P Systems llevado a cabo por M.A. Gutiérrez Naranjo y D. Ramírez Martínez, de la Universidad de Sevilla. Esta herramienta software permite trabajar con SN P Systems en un modo de paso a paso. Para ello, el sistema cuenta con la posibilidad de introducir el diagrama que se quiere simular de una forma gráfica e interactiva [8].

3.4 Membrane Computing Simulator (MeCoSim)

La herramienta MeCoSim (Membrane Computing Simulator[9]), es un software que ofrece a los usuarios una Aplicación de Propósito General (General Purpose Application) para modelizar, simular, analizar y verificar diferentes tipos de modelos basados en P Systems. Algunas de las posibilidades de esta herramienta son las siguientes:

- Simulación de modelos de P Systems bajo diferentes condiciones iniciales. También posibilita la carga de modelos basados en P-Lingua, parsing, edición, debugging y diferentes tipos de simulación.

- Visualización de las componentes de análisis: alfabeto, membrana, estructura, multiconjuntos e interfaz gráfica.

- Plataforma altamente configurable para la definición de los valores de entrada y salida, parámetros y diseños gráficos para cada modelo o familias de P Systems.

- Sistema de repositorios para la administración visual con diferentes recursos online, incluyendo plugins, aplicaciones personalizadas, modelos y escenarios.

- Opción de exportación para la resolución de problemas prácticos.

3.5 MP Virtual Laboratory

Por otro lado, encontramos la herramienta MP Virtual Laboratory [10], orientada a la asistencia en la creación de modelos para entender los mecanismos biológicos internos de los sistemas y para predecir la respuesta a estímulos externos, modificaciones de las condiciones ambientales, o cambios en la estructura. La interfaz de MP Virtual Laboratory es capaz de diseñar y controlar modelos biológicos, y un extenso conjunto de plugins a través de los cuales pueden ser generados y analizados estos tipos de sistemas.

En relación con el caso que nos interesa en este apartado, se analizan los denominados Metabloc P Systems (MP Systems), como una especie de clase determinista de P System, propuesta como modelo de metabolismos biológicos. Sus dinámicas se computan con conjuntos molares de reescritura regulados por funciones, donde las reacciones tienen lugar en el objeto o en las poblaciones. La dinámica es determinista a nivel de población, donde nada se puede decir sobre la evolución dinámica de objetos únicos.Esta situación se asemeja a lo que sucede en las leyes de los gases macroscópicas, que especifican las relaciones entre determinadas medidas de presión, volumen y temperatura, pero no hacen frente al comportamiento mecánico de las moléculas individuales.

Como conclusión a todos los apartados anteriores, ninguna de las herramientas descritas consigue llevar a cabo los objetivos que se han planteado anteriormente en el apartado 1.2 de una manera sencilla y estrictamente dedicada a los Spiking Neural P Systems, salvo el simulador desarrollado por Gutierrez Naranjo y Ramírez Martínez. Sin embargo, su implementación no es accesible mediante Web y se encuentra un tanto desactualizada. Por ello, se ha decidido la creación de una herramienta propia accesible a cualquier usuario vía web, utilizando un lenguaje de desarrollo potente y robusto como R.

4 Aplicaciones de los SN P Systems especialmente consideradas en la simulación

Los SN P Systems son un modelo de computación versátil que puede ser usado para diseñar algoritmos paralelos muy eficientes que resuelven conocidos problemas de la ciencia de la computación. A continuación se va a estudiar la capacidad de estos sistemas para simular circuitos booleanos ya que, aparte de ser un bien conocido modelo computacional, existen numerosos algoritmos que resuelven problemas en este campo. Además, si a esta simulación se le dota de módulos de memoria, pueden constituir una prueba alternativa de la completitud computacional del modelo.

Otra cuestión de estudio se refiere a la ordenación de un vector de números naturales utilizando sistemas SN P Systems. En este caso, debido a sus características paralelas, la complejidad de tiempo obtenida por el algoritmo propuesto supera los resultados clásicos. Varios problemas abiertos surgieron durante la investigación en estas dos áreas. Por ejemplo, en el caso de los circuitos booleanos, la simulación se realiza para aquellos circuitos cuyos diseños se presentan en forma de árbol, por lo que aún queda el estudio de la simulación de diferentes tipos de circuitos a través del uso de SN P Systems. En lo que se refiere al algoritmo de ordenación, la construcción presentada depende de la magnitud de los números a ser ordenados. Se conjetura que este inconveniente puede ser eliminado. Además, se conjetura que se pueden hacer más mejoras relativas a la complejidad de tiempo.

4.1 Puertas y Circuitos Lógicos

En primer lugar, definiremos la estructura y la semántica que se va a emplear más adelante en el estudio de este caso, basado en el análisis que se plantea en [2].

Una función booleana n-aria es una función $f\{true, false\}^n \rightarrow \{true, false\}$. Hay tres funciones booleanas primarias que son ampliamente utilizadas:

- **NOT**. Esta función es simplemente una negación, es decir, la salida es la opuesta a la entrada. La función NOT toma sólo un valor de entrada, por lo que se le denomina función unaria u operador. La salida es $true$cuando la entrada es $false$, y viceversa.
- **AND**. Esta función toma el valor $true$ a la salida sólo en el caso en el que todas las entradas sean $true$, si no, devolverá $false$.
- **OR**. El resultado de esta función será $true$ si existe alguna entrada con valor $true$.

Ambas funciones, AND y OR, pueden tomar cualquier número de entradas, siendo éste como mínimo dos.

Existe potencialmente una forma más económica de representar funciones booleanas, en forma de circuitos booleanos. Un circuito booleano es un grafo $C = (V, E)$, donde los nodos en $V = \{1, \ldots, n\}$ son las puertas lógicas de C. El grafo C presenta una estructura concreta. En primer lugar, no existen ciclos en él, por lo que podemos asumir que todas

las conexiones entre nodos son de la forma (i, j) con $i < j$. Todos los nodos en el grafo tienen el "indegree" (número de conexiones entrantes) igual a 0, 1 ó 2.

También, cada puerta $i \in V$ tiene un clase $s(i)$ asociada a ella, donde

$$s(i) \in \{true, false, OR, NOT, AND\} \cup \{x_1, \dots, x_n\}.$$

- Si $s(i) \in \{true, false\} \cup \{x_1, \dots, x_n\}$, el indegree de i es 0, es decir, i no debe tener ninguna conexión entrante. A este tipo de puertas se les denomina *inputs* de C.

- Si $s(i) = NOT$, entonces i tiene *indegree* igual a 1.

- Si $s(i) \in \{OR, NOT\}$, el indegree ha de ser dos obligatoriamente. Finalmente, el nodo n no puede tener conexiones con otros nodos, y es la denominada puerta de salida del circuito.

Esto concluye la definición de la sintaxis de los circuitos y, a continuación, estudiaremos la semántica. Sea $X(C)$ el conjunto de todas las variables booleanas que aparecen en el circuito C (esto es, $X(C) = \{x \in X\}$; $s(i) = x$ para alguna puerta i de C). Diremos que unos valores de verdad T son apropiados para C si está definido para todas las variables en $X(C)$. Dado T, el valor de verdad de una puerta $i \in V$, $T(i)$, es definido, por inducción en i, como sigue: $T(i) = T(s(i))$.

- Si $s(i) = NOT$, hay una única puerta $j < i$ tal que $(j, i) \in E$. Por inducción, sabemos $T(j)$, y entonces $T(i)$ es $true$ is $T(j)$ es $false$, y viceversa.

- Si $s(i) = OR$, entonces hay dos conexiones (j, i) y (j', i) que conectan con i. $T(i)$ será $true$ si alguna de los $T(j), T(j')$ es $true$.

- Si $s(i) = AND$, entonces $T(i)$ será $true$ si ambos valores, $T(j)$ y $T(j')$ son $true$. Finalmente, el valor del circuito, $T(C)$, será $T(n)$, donde n representa a la puerta de salida.

4.1.1 Simulación de puertas lógicas

En este apartado analizaremos cómo los SN P Systems pueden simular puertas lógicas. Consideramos que la entrada se muestra en una neurona mientras que la salida será tomada de la neurona de salida del sistema. Por otro lado, el valor lógico 1 será codificado como dos spikes, aa, mientras que el 0 se representará con un único spike, es decir, que si la neurona de salida emite dos spikes en el segundo paso de la computación, el valor booleano computado será el 1, $true$. Si sólo emite 1 spike, el resultado será 0, $false$.

A continuación se ofrecerá una serie de resultados a tener en cuenta sobre la construcción de puertas lógicas con SN P Systems:

- *La puerta lógica AND puede ser simulada por un SN P System usando una neurona y sin retardo en las reglas, todo ello en un ciclo de computación*.

Construiremos el siguiente SN P System: $\Pi_{AND} = (\{a\}, \sigma_1 = (0, a^2 \rightarrow a;\ 0, a^3 \rightarrow a;\ 0, a^4/a^2 \rightarrow a; 0),\ \emptyset, 1)$, que se muestra en la Figura 6. El funcionamiento del sistema será de la siguiente forma: suponiendo que se introducen 3 spikes en la neurona 1, lo que significa que queremos calcular la operación AND entre los valores lógicos 1 y 0, o 0 y 1. La única regla que el sistema puede usar es $a^3 \rightarrow a; 0$, por lo que se ejecuta dicha regla y se emite en la salida un spike, dando lugar al valor lógico 0.

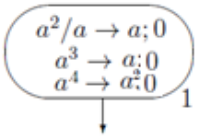


Figura 6 Un SN P System simulando la puerta AND

Si se introducen 4 spikes en la neurona 1, estaremos realizando el cómputo lógico 1 AND 1 y el sistema ejecutará la regla $a^4 \rightarrow a^2; 0$, por lo que la neurona 1 emitirá 2 spikes. El sistema con los valores 00 como entrada se comportará de forma similar a los casos 0 AND 1 y 1 AND 0. Por tanto, hemos mostrado cómo el sistema construido devuelve las salidas correctas en un único paso de computación, volviendo a la configuración inicial para su uso posterior en caso de ser necesario.

- *La puerta lógica OR puede ser simulada por un SN P System usando una neurona y sin retardo en las reglas, en un ciclo de computación*.

  Este SN P System tiene la siguiente estructura y se muestra en la figura 7:

$$\Pi_{OR} = (\{a\}, \sigma_1 = (\{a^2 \rightarrow a; 0, a^3 \rightarrow a^2; 0, a^4 \rightarrow a^2; 0\}), \emptyset, 1)$$

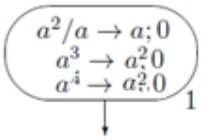


Figura 7 Un SN P System simulando la puerta OR

- *La puerta lógica NOT puede ser simulada por un SN P System usando dos neuronas, sin retardo en las reglas, en dos pasos*.

Para su comprobación, construiremos el siguiente SN P System: $\Pi_{NOT} = (\{a\}, \sigma_1, \sigma_2, \{(1,2), (2,1)\}, 1), y\ \sigma_1 = (a, \{a^2 \rightarrow a^2; 0, a^3 \rightarrow a; 0\}), \sigma_2 = (0, \{a/a \rightarrow a; 0,\ a^2/a^2 \rightarrow a; 0\})$.

En la configuración inicial del sistema, la neurona 1 contendrá 1 spike, el cual, una vez sea usado para la simulación de la puerta, debe ser restaurado para que el sistema vuelva al estado inicial y pueda comenzar desde el mismo estado del que partió. Esto ocurre gracias a la ayuda de la neurona 2, la cual, en el segundo ciclo de ejecución, emite a la neurona 1 el spike que le falta.

En la Figura 8se muestra la representación gráfica del SN P System anterior. El funcionamiento será como sigue: si la entrada vale 1, a la neurona 1 se le añadirán 2 spikes. Teniendo 3 spikes, dos insertados por la entrada y uno que ya contenía, la neurona 1 sólo puede usar la regla $a^3 \rightarrow a; 0$, por lo que emitirá a la salida un spike, lo que quiere decir, que la evaluación de la puerta lógica es false, habiéndose realizado el funcionamiento correcto de la puerta NOT. Además de emitir a la salida un spike, enviará otro a la neurona 2, con la que presenta conexión. En el siguiente ciclo de ejecución, la neurona 2 por el uso de la regla $a \rightarrow a; 0$ hace que el sistema vuelva a su estado inicial.

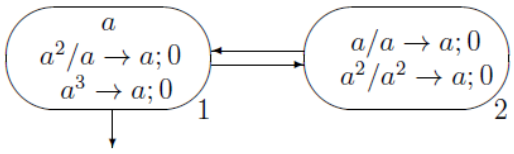


Figura 8 Un SN P System simulando la puerta NOT

En el caso en el que la entrada al sistema fuera 0, un spike se introduce en la neurona 1, usará la regla $a^2 \rightarrow a^2$, dos spikes se envían a la salida y a la neurona 2 al mismo tiempo. En la salida se procesará el 1 lógico, dos spikes, y en la neurona 2 se aplicará la regla $a^2 \rightarrow a; 0$, por lo que consumirá los dos spikes y enviará uno de vuelta a la neurona 1, volviendo a encontrarse en el estado inicial.

Una vez que se ha mostrado el uso y la simulación se las puertas lógicas AND, OR y NOT, estudiamos el caso de los circuitos lógicos.

4.1.2 Simulación de circuitos lógicos

A continuación, se presenta un ejemplo de cómo construir un SN P System para simular un circuito lógico que evalúa una función booleana. Para ello, se hará uso de los sistemas creados anteriormente para cada una de las puertas NOT, AND y OR, a los que se añadirán neuronas extra para sincronizar el sistema de forma que se llegue a una salida correcta.

Comenzamos con la implementación del circuito dado por la siguiente función lógica:$f(x_1, x_2, x_3, x_4) = (x_1 \wedge x_2) \vee \neg(x_3 \wedge x_4)$. El circuito correspondiente se muestra en la Figura 9, descrito en [2].

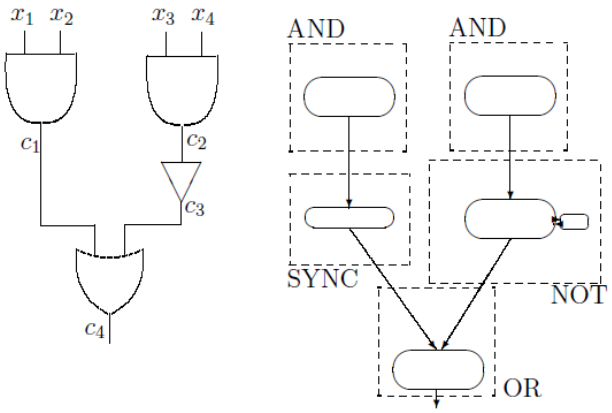


Figura 9 Circuito lógico y su SN P System asociado

Para que el SN P System que simula el circuito muestre en la salida el resultado correcto, es necesario para cada sub-sistema que simula las puertas lógicas descritas anteriormente, AND, OR, NOT, que reciba la entrada de las salidas anteriores al mismo tiempo para niveles iguales de profundidad de las puertas lógicas. Para lograrlo, hemos de añadir neuronas de sincronización, inicialmente vacías con una única regla de la forma $a \rightarrow a; 0$.

Para llevar a cabo la simulación completa del circuito formado por sub-sistemas AND, NOT y OR, procederemos de la siguiente forma:

- Para cada puerta del circuito con inputs de las entradas de las puertas, tendremos un SN P System que lo simule. La entrada se da en la neurona numerada como 1 de cada puerta. Esto se puede ver reflejado en el sistema de la derecha de la Figura 9, donde el primer nivel está formado por los SN P Systems que simulan el comportamiento de las dos puertas AND. En el segundo nivel, vemos el SN P System para la puerta NOT, y por último, en el tercer nivel, el sistema que simula el funcionamiento de la puerta OR.
- Para cada puerta que tenga al menos una entrada proveniente de la salida de una puerta lógica anterior, construiremos un SN P System que lo simule construyendo una conexión entre la neurona de salida de la puerta primera, con la entrada al sub-sistema que simula el funcionamiento de la puerta posterior. En este caso, en la Figura 9, se añade un sistema de sincronización SYNC que impide que la salida de la primea puerta AND llegue a la puerta OR antes de que se haya generado la salida de la puerta NOT ubicado en el mismo nivel que este módulo de sincronización. De esta forma, las entradas de la puerta OR será la correcta y generará la salida correcta.

Ha de tenerse en cuenta que si la sincronización es necesaria, una nueva conexión se realiza entre la salida de la neurona de la puerta de salida y la neurona de sincronización. A continuación, ésta se une con la entrada de la nueva puerta del circuito que se desea construir.

Generalizando las observaciones que se han realizado, se obtiene lo siguiente:

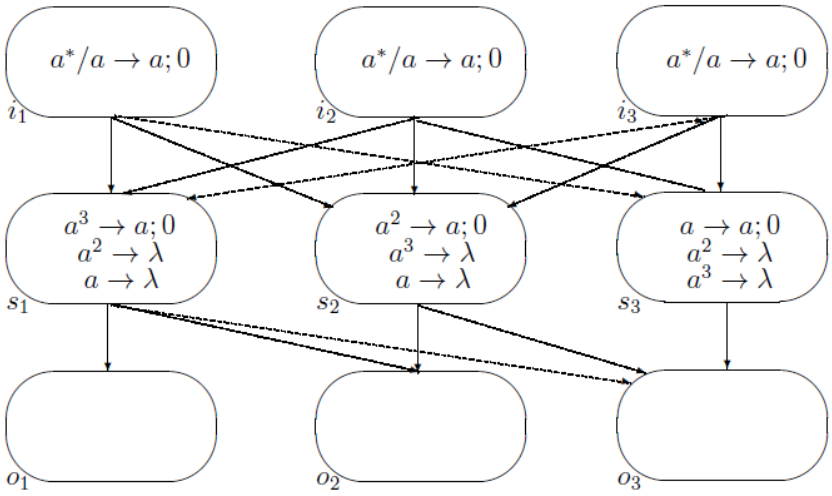
- Todo circuito α, cuya estructura es de tipo árbol como la explicada anteriormente puede ser simulado con un Spiking Neural P System, Π_α, en tiempo lineal. Π_α se construye haciendo uso de los SN P Systems del tipo Π_{AND}, Π_{OR} y Π_{NOT}, reproduciendo en la arquitectura de los sistemas, la estructura del árbol asociado al circuito.

4.2 Algoritmo de Ordenación de números naturales

Pasemos ahora a un problema diferente que un SN P System puede resolver: ordenar n números naturales. En primer lugar, explicaremos cómo será el esquema y funcionamiento de este sistema a través de un ejemplo expuesto en [2].

Supongamos que se quiere ordenar tres números naturales, 1, 4 y 2, en este orden concreto. Construyamos el SN P System que se aparece en la F*i*gura 10.

La entrada del sistema serán los números a ordenar codificados como cantidad de spikes, es decir, las neuronas del primer nivel, i_1, i_2, i_3 contendrán 1, 4 y 2 spikes, respectivamente. Se puede observar que las neuronas de entrada contienen todas la misma regla $a^*/a \to a; 0$ y presentan conexión con todas las neuronas del segundo nivel, s_1, s_2, s_3. La neurona identificada como s_1 tiene conexión con todas las neuronas del tercer nivel del sistema, contiene una regla de la forma $a^3 \to a; 0$, donde 3 es la cantidad de números a ordenar, y dos reglas de eliminación, $a^2 \to \lambda$ y $a \to \lambda$. Para las otras neuronas del segundo nivel, el exponente de las reglas que emiten spikes decrece en uno para cada neurona de dicho nivel, al igual que las conexiones con las neuronas del tercer nivel, identificadas como o_1, o_2, o_3.



F*i*gura 10 SN P System que ordena tres números naturales

En la configuración inicial del sistema tendemos un spike en la neurona i_1, cuatro en la neurona i_2 y dos en la neurona i_3. En el primer paso de la ejecución, un spike de cada neurona se consume y se envía a las neuronas del segundo nivel del sistema. Cada una de ellas recibe el mismo número de spikes, 3.

En el segundo paso de la ejecución, la neurona s_1 consume sus tres spikes recibidos en el paso anterior y envía uno a las neuronas o_1, o_2, o_3. Ahora, cada neurona de la línea de salida contiene un spike. Las otras neuronas del segundo nivel eliminan los tres spikes que habían recibido. En el mismo ciclo, las neuronas i_2 y i_3 emiten dos spikes cada una a todas las neuronas del segundo nivel.

En el tercer paso de la ejecución, la neurona s_2 envía sólo a las neuronas o_2 y o_3(por lo que éstas tendrán un spike más, 2, mientras que o_1 sólo contendrá 1), los otros spikes de las neuronas s_1 y s_3 se eliminan. En el mismo tiempo, la neurona i_2 añade un spike a cada una de las neuronas del segundo nivel, los cuales serán consumidos en el cuarto paso de la computación por la neurona s_3 y enviados a la neurona de salida o_3. El último ciclo es equivalente al cuarto y otro spike se envía a la neurona o_3.

Por lo tanto, en el último paso hay 1 spike en la neurona o_1, 2 spikes en la neurona o_2, y 4 en la neurona o_3, tal y como esperábamos.

Pasamos ahora al caso general, construyendo el esquema del sistema que se muestra en la Figura 11.

El funcionamiento del sistema es simular a lo explicado en el ejemplo anterior. Se introducen n números naturales codificados como cantidad de spikes, en cada una de las neuronas del primer nivel del SN P System (denotadas por i_j, con $1 \leq j \leq n$). Hasta que no contengan ningún spike, consumen uno en cada ciclo y envían n spikes, uno para cada neurona del segundo nivel (denotadas por s_i, con $1 \leq i \leq n$). Estas neuronas presentan una regla de la forma $a^h \rightarrow a; 0$, donde h tiene el valor n en la neurona s_1, y va decreciendo una unidad hasta valer 1 en la neurona s_n. También presentan n diferentes conexiones con las neuronas del tercer nivel del sistema. Éstas últimas serán las encargadas de almacenar el resultado de la computación.

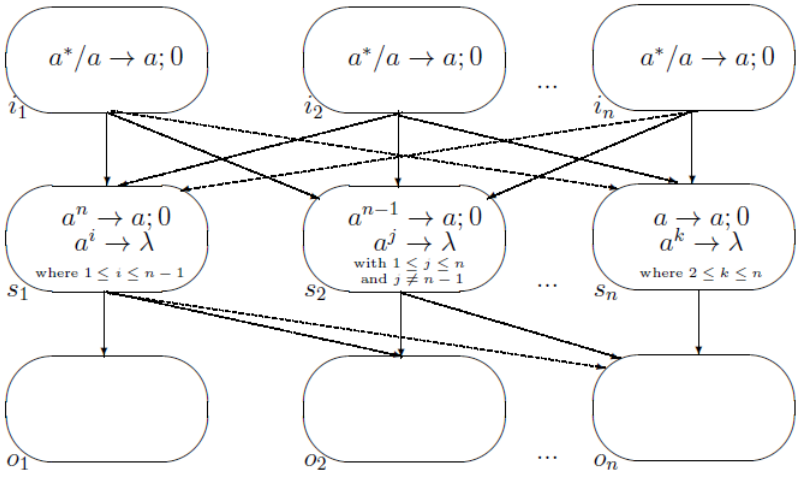


Figura 11 SN P System que ordena n números naturales

Por lo explicado anteriormente, figura en [2] el siguiente teorema:

- *Los SN P Systems pueden ordenar un vector de números naturales que recibe el sistema, codificados como cantidad de spikes, llevando a cabo la tarea de ordenación en* $t + 1$ *ciclos de computación, donde* t *es el máximo de los números que se desean ordenar.*

En función de la construcción descrita, el tiempo de ordenación de los números será $O(T)$, entendiendo tiempo de ordenación como el número de configuraciones que ha llevado a cabo el sistema, y T, como la magnitud de los números a ordenar. Aunque este orden pueda ser mejor que el de otros algoritmos habituales, hemos de tener en cuenta que la construcción depende el número de elementos a ordenar.

5 Análisis

Una vez que se han establecido los fundamentos teóricos y se han analizado algunas aplicaciones de interés especial en este trabajo, a lo largo de este capítulo se determinarán las necesidades y las funciones que debe satisfacer el simulador, teniendo en cuenta los diversos fundamentos expuestos anteriormente.

5.1 Descripción del sistema

5.1.1 Contexto

El sistema a desarrollar está dirigido a cualquier usuario que quiera simular los diferentes estados de la computación de un SN P System, facilitando el trabajo con éstos, de forma que sea lo más sencillo posible obtener los resultados deseados. Además, se quiere reducir el tiempo que se utiliza en resolver este tipo de simulaciones de forma manual, o a través del diseño de herramientas propias. El sistema debe permitir llevar a cabo la evaluación del estado del SN P System en cada ciclo de ejecución y dar la posibilidad al usuario de avanzar en su simulación de acuerdo a las reglas que le interesen aplicar.

5.1.2 Usuarios del sistema

El simulador a desarrollar no tiene usuarios diferenciados, por lo que no es necesario autenticarse en la aplicación para poder trabajar con ella.

5.2 Definición del sistema

5.2.1 Modelado de Casos de Uso

A continuación, se muestra la figura con los casos de uso definidos para el sistema.

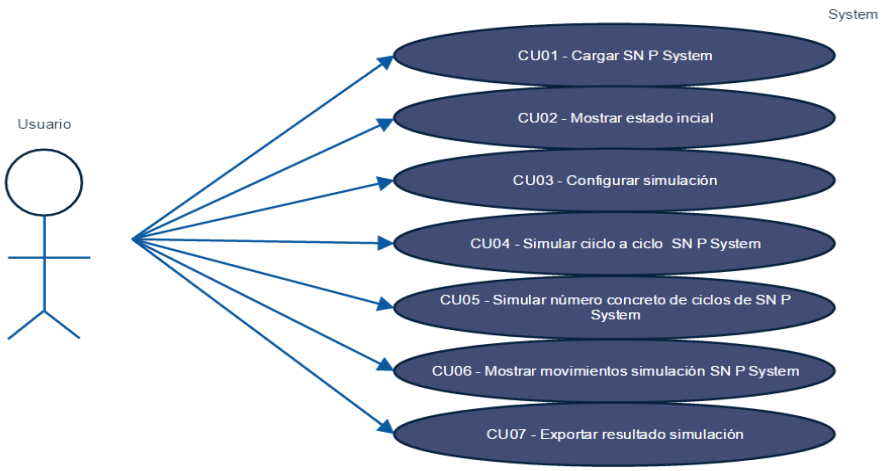


Figura 12 Diagrama de Casos de Uso

Como se aprecia en la Figura 12, para cumplir con los objetivos del sistema el simulador debe permitir:

- **Cargar un SN P System.** Para proceder a la simulación se cargará la información desde tres ficheros: el de spikes, el de reglas y el de conexiones.

- **Mostrar el estado inicial del SN P System**. Para que el usuario puede comprobar que la carga ha sido exitosa y corresponde con lo esperado, se mostrará un diagrama gráfico y las tablas que representen la información cargada en la herramienta.

- **Configurar simulación**. Para poder simular un SN P System se debe indicar el tipo de sistema (normal, *total spiking* o *decaying spikes*), el tipo de salida de la simulación (ciclos transcurridos entre dos emisiones de spikes de la neurona de salida, o número de spikes emitidos por ésta en cierto ciclo de la ejecución), y si se desea exportar la información de la simulación o no.

- **Simular el sistema ciclo a ciclo**. Para ello, el usuario deberá indicar qué regla desea aplicar cuando exista más de una posible para ser ejecutada en una neurona en un ciclo determinado.

- **Simular el sistema para un número de ciclos especificado**. Para ello, el usuario no necesita realizar ninguna función adicional.

- **Mostrar los movimientos que son consecuencia de la simulación realizada**. De esta forma el usuario conocerá exactamente cada uno de los pasos intermedios que han dado lugar al resultado final obtenido.

- **Generar la exportación** de todos los datos de la simulación, permitiéndose la carga del SN P System para una posterior continuación de la simulación.

5.2.2 Requisitos Funcionales

A partir de los Casos de Uso definidos en el diagrama anterior y del funcionamiento descrito en el apartado de Fundamentos Teóricos, se realiza la especificación detallada de los casos de uso, cuyo contenido puede verse con detalle en el Anexo A.

5.2.3 Requisitos No Funcionales

RNF 1. El sistema debe ser capaz de procesar y mostrar la salida en un tiempo máximo de 2 segundos, siendo preferible, 1 segundo o menos.

RNF 2. El número máximo de neuronas que el sistema puede manejar estará comprendido entre 1 y 10.

6 Diseño

En este capítulo se presenta la información de diseño del simulador desarrollado. En primer lugar se analiza la herramienta de desarrollo y posteriormente se proporciona el diagrama de clases. Se incluye también la descripción de la arquitectura del sistema y detalles sobre la interfaz y el diagrama de estados.

6.1 Herramientas de desarrollo

Para llevar a cabo el desarrollo del simulador de SN P Systems, se ha utilizado el lenguaje R, debido a diferentes razones:

- **Robustez.** Es un lenguaje robusto con una curva de aprendizaje compleja, pero muy efectivo para el manejo de datos estadísticos. Es un lenguaje orientado a objetos, con una sintaxis similar a la de C y C++.

- **Facilidad en la preparación de los datos.** Cuando se manejan volúmenes de datos muy elevados, gran parte del tiempo se dedica a preparar la información para la visualización de la que se pueden extraer conclusiones. Con R esa preparación es relativamente sencilla, en gran medida porque automatiza muchos procesos mediante la programación de *scripts*.

- **R funciona con cualquier tipo de archivo.** R es muy flexible, puede trabajar con datos procedentes de distintas fuentes: txt, csv, JSON o xls.

- **Visualizaciones.** Si hay algo que define a R es su capacidad para visualizar información compleja de una forma sencilla. Dispone de librerías especializadas en visualización de datos para mostrar gráficos.

Debido a que en gran medida estas características son las que se buscaban para el simulador, se ha optado por utilizar este lenguaje a través del entorno de desarrollo integrado, denominado RStudio [11].

6.2 Diagrama de clases

La Figura 13 resume el Diagrama de Clases para el simulador. A continuación, se explica cada una de las clases:

- **Spikes**: Será la encargada de almacenar toda la información relativa a la cada uno de los spikes que contiene el sistema. La variable *cellID* indicará la célula que posee el spike en ese momento, *actualTimeOfLife* indicará el número de ciclos que lleve el spike en la neurona, *maxTimeOfLife* indicará, en el caso de que la configuración del sistema sea de decaying spikes, el número máximo de ciclos que dicho spike permanecerá en esa célula sin ser usado antes de ser eliminado. Una vez haya transcurrido dicho número de ciclos, éste pasará al estado inactivo, y a efectos de su utilización, será como si no figurara en el sistema. Por último, *active* indicará si el spike está presenta en la citada célula o si ya ha sido consumido o ha sido eliminado por el paso del tiempo.

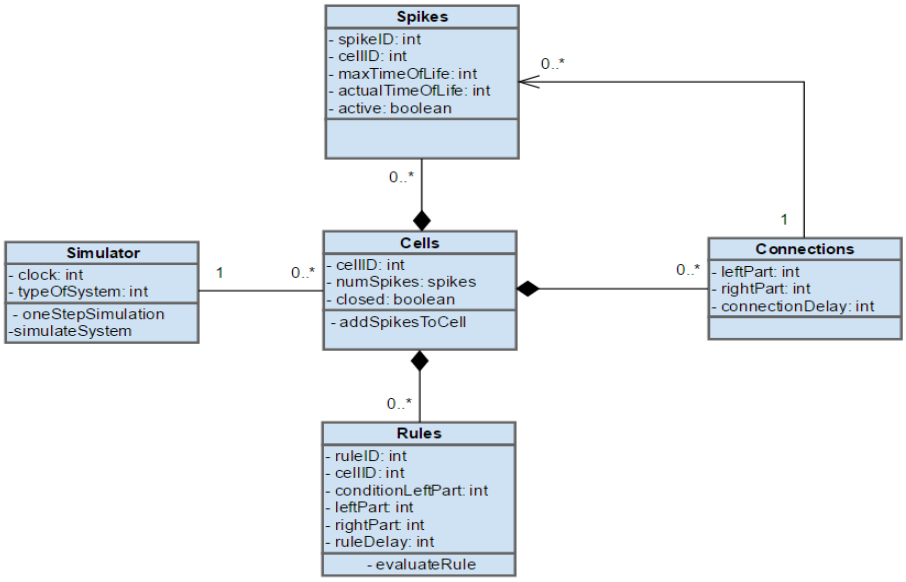


Figura 13 Diagrama de clases del sistema

- **Cells**: Esta clase podríamos decir que es auxiliar, puesto que guarda información redundante, y se usa para agilizar las operaciones y las comprobaciones de las reglas. La variable *numSpikes* indica el número de spikes que contiene cada una de las células de tal forma que para comprobar si una regla puede ser usada o no, sólo es necesario comprobar con esta tabla y no comprobar con la tabla que contiene todos los spikes de todas las células del sistema. Por este hecho, cada vez que finaliza un ciclo de ejecución, se actualiza comprobando todas los spikes activos del sistema e indicando el número total existente en cada célula en dicho ciclo. La variable *closed* indica si la neurona correspondiente a *cellID* ha ejecutado una regla con retardo y no permite ni recibir ni emitir spikes.

- **Connections:** Esta clase indica las conexiones que existen entre las células (neuronas) del sistema. Si la célula *i* está conectada con la *j*, *leftPart* contendrá el *cellID* de la célula *i*, y *rightPart* el cellID de la célula *j*. La variable *connectionDelay* indica el número de ciclos de reloj que han de transcurrir hasta que el spike llegue a la célula correspondiente desde que es emitido debido a la ejecución del movimiento. Para indicar la célula destino, la identificaremos con el cellID -1.

- **Rules:** Esta clase contiene toda la información relativa a cada una de las reglas que componen las células del sistema. La variable *cellID* representa el identificador de la célula a la que pertenece la regla, *conditionLeftPart* indica el número de spikes que ha de tener la neurona para poder ejecutar esa regla, *leftPart* indica la cantidad de spikes que se consumirán al ejecutar la regla, *rightPart* contiene la cantidad de spikes que van a ser emitidos a las células con las que existe conexión cuando se ejecute la citada regla y, por último, *ruleDelay* indicará la cantidad de ciclos que han de transcurrir hasta que el movimiento sea realmente ejecutado. En el ínterin de tiempo hasta que esto ocurra, la célula quedará en estado cerrado y no podrá recibir ningún spike, ni por supuesto, emitir otros. Cuando hayan pasado esos ciclos de reloj, la célula volverá a estar disponible para la recepción y envío de spikes.

Las características de R permiten hacer una sencilla analogía entre las clases y los elementos que R denomina *dataframes*. Éstos se usan para almacenar tablas de datos y son una lista de vectores de igual longitud. En el caso que nos ocupa, identificaríamos cada una de las filas que forman los *dataframes* con cada una de las instancias de una clase. A su vez, los métodos se asemejarían a las funciones definidas en R que trabajan con las tablas creadas.

6.3 Descripción de la arquitectura del sistema

La Figura 14 muestra la arquitectura modular básica del simulador.

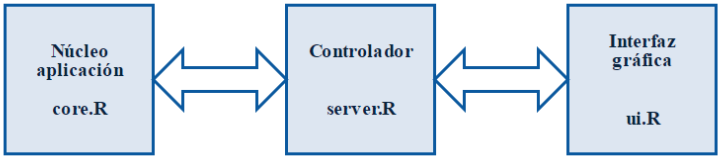


Figura 14 Arquitectura del simulador

El *núcleo* de la aplicación estará formado por todas las funciones diseñadas en R que permiten el funcionamiento lógico del simulador, es decir, llevar a cabo la computación de los SN P System. A continuación se detallan las funcionalidades más importantes:

- **addSpikeToCell:** Esta función será la encargada de añadir un spike a la célula que se indique. Para ello, recibe el tiempo de vida máximo y la célula a la que va a pertenecer. Con esta información, se añade en la tabla de spikes y formará parte de los spikes del sistema.

- **evaluateLeftPartRules:** Para llevar a cabo la evaluación se comprueba si la célula pose los spikes necesarios que indican la condición de la parte izquierda de la regla y, si esto es posible, se refleja este hecho en la tabla de movimientos. Como se ha indicado el capítulo de Fundamentos teóricos, esta selección se hace de una manera no determinista, por lo que la ejecución de la misma regla bajo las mismas condiciones varias veces puede ofrecer resultados diferentes. Por lo tanto, una vez que se ha seleccionado las reglas que se pueden aplicar, se devuelven todas aquéllas cuya ejecución fuera posible.

- **evaluateMovements:** Esta función es la encargada de la evaluación de los movimientos que van a tener lugar en el sistema. Entendiendo un movimiento por el resultado a aplicar tras la ejecución efectiva de una regla, éste dará lugar a la generación de una serie de spikes y al consumo de otra cantidad de la célula que contiene la citada regla. Será posible evaluar cada una de las entradas de la tabla de movimientos cuando el sistema se encuentre en el ciclo de ejecución que indica el movimiento. Cuando esto ocurra, se actualizarán los spikes que contiene la célula destino, junto con su tiempo de vida máximo si procede, y se consumirán de la célula que ha provocado el movimiento la cantidad de spikes que indica la regla

ejecutada. Una vez realizado esto, se marcará dicho movimiento como ejecutado y no se volverá a usar en un futuro.

- **oneStepSimulation:** Esta función es la encargada de realizar un paso en la simulación. Consta de las siguientes acciones, por orden de ejecución:

    1. En primer lugar se comprueban las reglas que pueden ser evaluadas para cada una de las neuronas en función de los spikes que contengan en tal instante d tiempo.

    2. A continuación, de una forma no determinista, se eligen las reglas a ser ejecutadas. También hay que tener encuentra que las células pueden encontrarse en estado "cerrado" si han lanzado spikes con retardo en tiempos anteriores y aún no han sido emitidos a las células conectadas con ésta. Por otro lado, los spikes generados por la regla aplicada se insertan en la tabla de movimientos para ser evaluados.

    3. Ahora llega el momento de la ejecución de los movimientos posibles, es decir, se comprueba que ha llegado el tiempo de ejecución correspondiente, habiendo tenido en cuenta los retardos de las reglas y de las conexiones entre células. Si esto es así, se generan los spikes y se asignan a las células correspondientes, sin olvidar eliminar los spikes consumidos que han producido la ejecución de las reglas.

    4. Después de haber evaluado los movimientos posibles, se comprueba el tiempo de vida de todos los spikes, si procede por el tipo de configuración empleada. Si el tiempo de vida ha llegado al máximo sin ser empleados en la célula que los contiene, se eliminan de ella y en el siguiente ciclo de ejecución ya no se tienen en cuenta para evaluar las reglas.

    5. Hecho todo lo anterior se aumenta en una unidad el reloj del sistema y se vuelve a comenzar con el procedimiento.

Para llevar a cabo la carga de los ficheros que contienen toda la información del SN P System, se necesitan tres ficheros que contengan:

- Los **spikes**. Este fichero tendrá cinco columnas, en las que se especificará el identificador del spike con un entero mayor que uno que no puede repetirse, la neurona a la que pertenece con un natural que es único y siempre el mismo para cada neurona, el tiempo de vida actual con un número mayor o igual que cero, su máximo tiempo de vida en el sistema con un número mayor o igual que cero, y si se encuentra activo o no con el valor *true* o *false*.

- Las **conexiones**. Este fichero tendrá tres columnas, en las que se especificará el identificador de la neurona de la que partirá el envío de un spike, el identificador de la neurona a la que llegará dicho spike, y el retardo que presenta esta conexión, mediante un entero mayor o igual que cero.

- Las **reglas**. Este fichero tendrá seis columnas, en las que se especificará el identificador único de la regla mediante un entero mayor que uno, el identificador de la neurona en la que se encuentra, y a continuación mediante un entero mayor o igual que cero los siguientes campos: la condición de la parte izquierda, la parte

izquierda de la regla, la parte derecha de la regla, y el retardo de aplicación de la regla.

6.4 Interfaz

A continuación se presentarán diferentes prototipos con el fin de mostrar cómo a través de la interfaz se accede a las diferentes funcionalidades del simulador y cómo ésta se encuentra organizada.

Desde la pantalla principal que aparece en la Figura 15 se tiene acceso al módulo de carga de la información del SN P System que se quiere simular y en la parte izquierda encontramos el menú general.

Para ello se introducirán los ficheros correspondientes a spikes, reglas y conexiones tal y como figuran en la Figura 16. Además, se puede incluir el nombre de la simulación para organizar los archivos en ese directorio como se puede comprobar en la Figura 17.

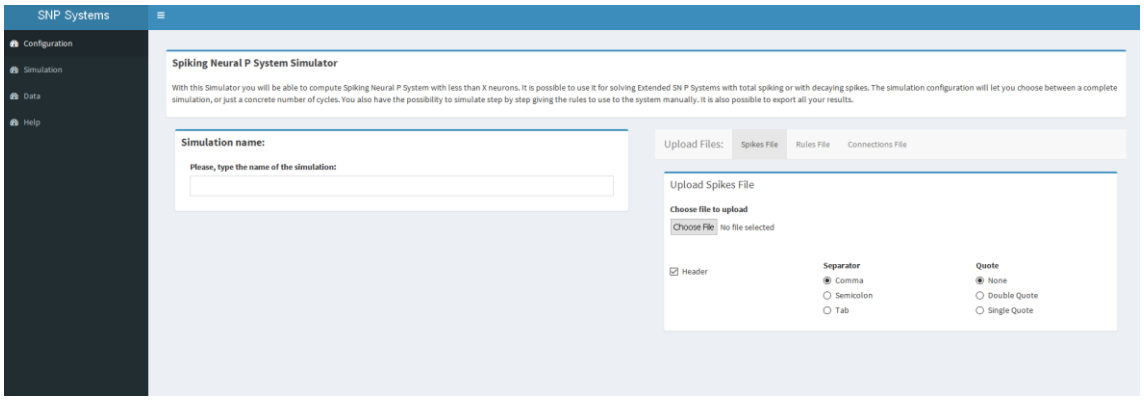


Figura 15 Pantalla principal de la aplicación

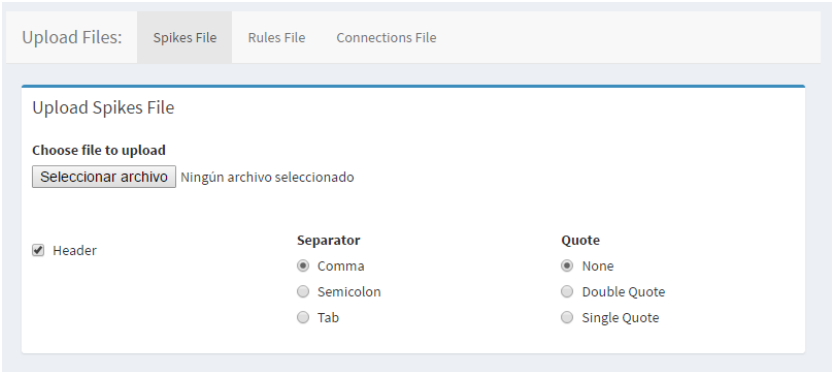


Figura 16 Zona de carga de la información del SN P System

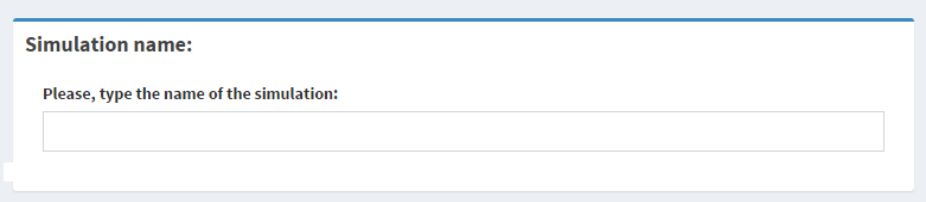


Figura 17 Área para introducir directorio de la simulación

Desde el menú izquierdo se puede acceder a la funcionalidad de la simulación, de consulta de la información cargada y a la ayuda de la aplicación.

Dentro del apartado de Simulación, encontraremos diferentes zonas: en la parte de ***EstadoInicial*** (Figura 18) se podrá comprobar el diagrama del SN P System cargado junto con las reglas que presentan cada una de las neuronas; en el apartado de configuración, se permitirá elegir el tipo de sistema a simular y configurar el resto de opciones; desde el apartado de ***EstadoFinal***, podremos ver el diagrama y los movimientos que han dado lugar a ese estado. Además, se permite desde esta zona la simulación paso a paso para ver la evolución del sistema según las reglas que el usuario desee aplicar cuando exista más de una posible para aplicar.

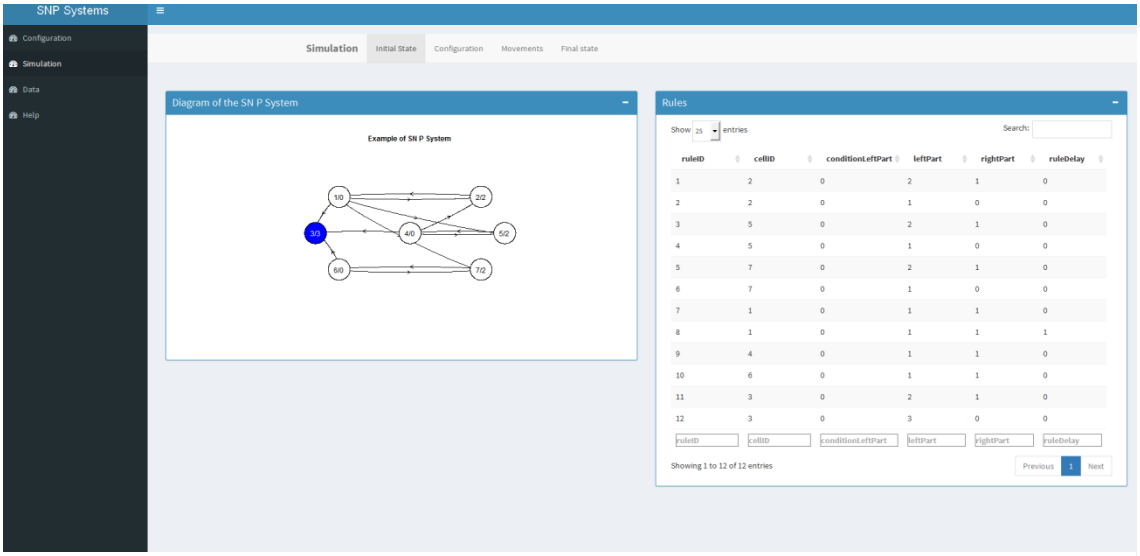


Figura 18 Pantalla principal de la opción de simulación

Desde el apartado de ***Data*** (Figura 19), podremos comprobar toda la información según ha sido cargada a través de los ficheros. Esto se podrá realizar en cualquier momento de la simulación.

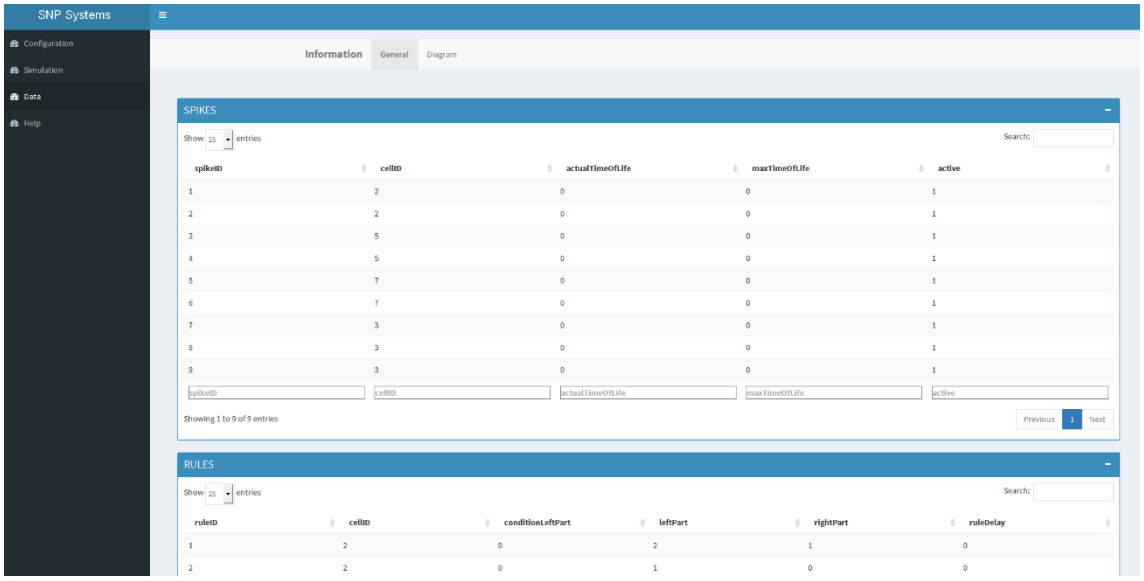


Figura 19 Pantalla que muestra la zona de información cargada

Desde el apartado de ***Help*** (Figura 20) encontraremos diferentes opciones de ayuda para llevar a cabo la simulación de un SN P System.

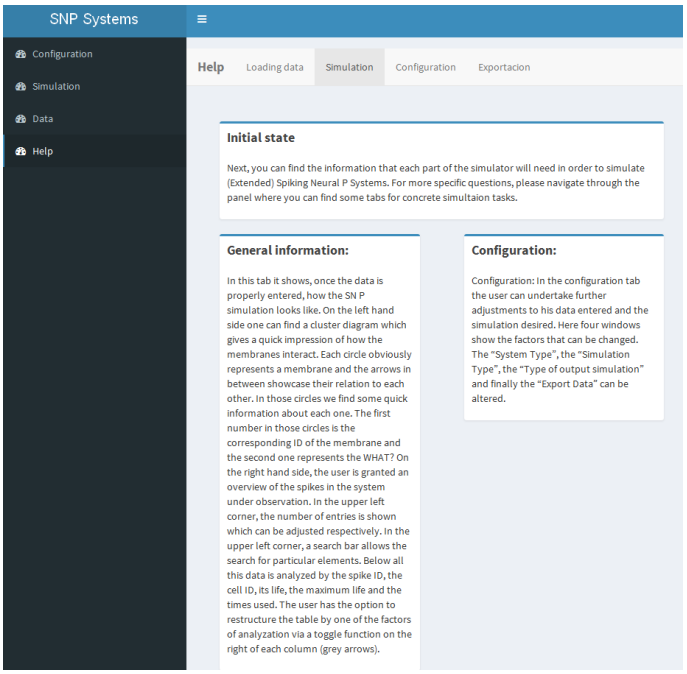


Figura 20 Pantalla que muestra la zona de ayuda

7 Pruebas y resultados

A lo largo de esta sección vamos a comprobar el funcionamiento del simulador mostrando los resultados de las pruebas que se han realizado:

7.1 Carga de los ficheros

La entrada al simulador está formada por tres ficheros: uno que contiene los spikes del sistema, otro que contiene las conexiones y otro las reglas de las neuronas (

Tabla 3, Tabla 4 y Tabla 5). El simulador toma esta información cuando el usuario introduce los diferentes archivos desde la pantalla inicial. Mediante Figura 21se muestra

• ruleID	cellID	conditionLeftPart	leftPart	rightPart	ruleDelay
1	2	0	2	1	0
2	2	0	1	0	0
3	5	0	2	1	0
4	5	0	1	0	0
5	7	0	2	1	0
6	7	0	1	0	0
7	1	0	1	1	0
8	1	0	1	1	1
9	4	0	1	1	0
10	6	0	1	1	0
11	3	0	2	1	0
12	3	0	3	0	0

la carga exitosa de los ficheros.

spikeID	cellID	actualTimeOfLife	maxTimeOfLife	active
1	2	0	0	TRUE
2	2	0	0	TRUE
3	5	0	0	TRUE
4	5	0	0	TRUE
5	7	0	0	TRUE
6	7	0	0	TRUE
7	3	0	0	TRUE
8	3	0	0	TRUE
9	3	0	0	TRUE

ruleID	cellID	conditionLeftPart	leftPart	rightPart	ruleDelay
1	2	0	2	1	0
2	2	0	1	0	0

3	5	0	2	1	0
4	5	0	1	0	0
5	7	0	2	1	0
6	7	0	1	0	0
7	1	0	1	1	0
8	1	0	1	1	1
9	4	0	1	1	0
10	6	0	1	1	0
11	3	0	2	1	0
12	3	0	3	0	0

Tabla 3 Fichero de entrada spikes.csv

Tabla 4 Fichero de entrada rules.csv

leftConnection	rightConnection	connectionDelay
2	1	0
5	4	0
7	6	0
1	2	0
1	5	0
1	7	0
1	3	0
4	2	0
4	5	0
4	3	0
6	7	0
6	3	0
3	-1	0

Tabla 5 Fichero de entrada connections.csv

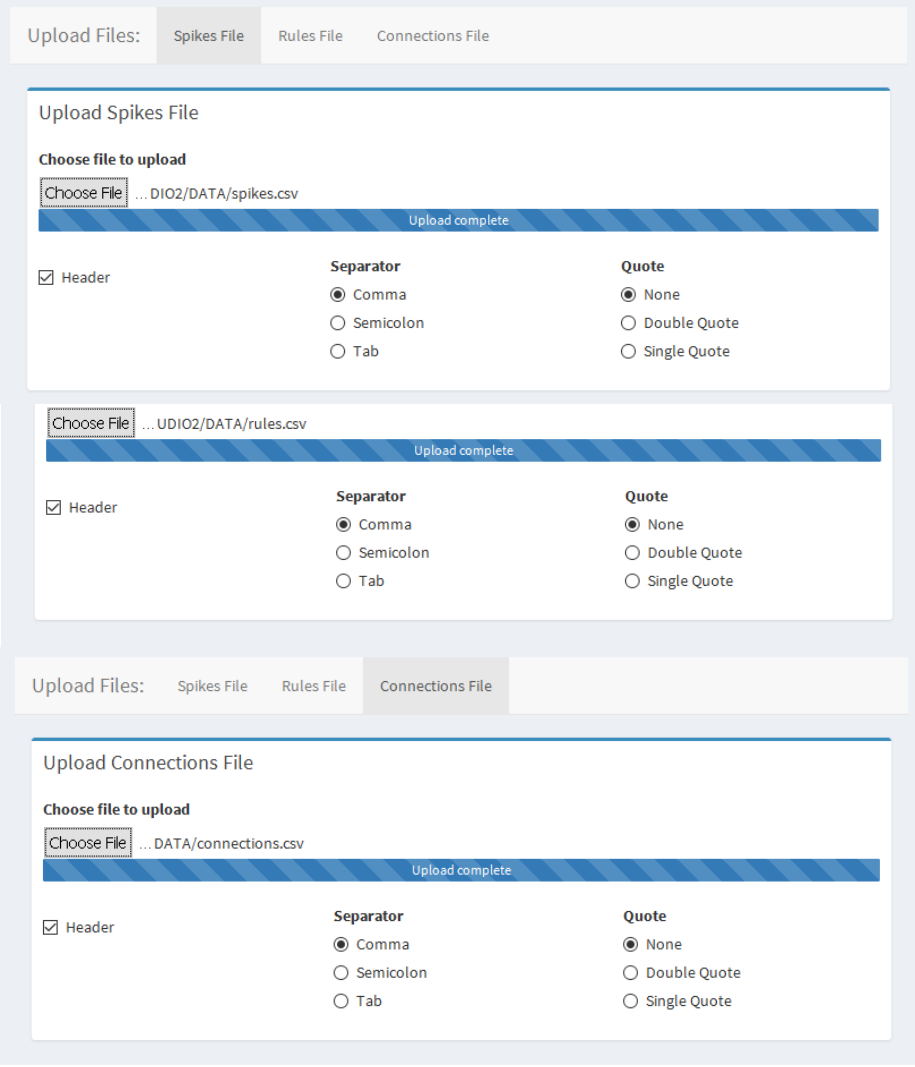


Figura 21 Capturas que muestran la carga exitosa de los ficheros

7.2 Vista preliminar

Una vez que se ha realizado la carga de los ficheros mencionados en el apartado anterior, se muestra la información almacenada de forma gráfica como muestra la Figura 22. Se pueden comprobar que ha generado un SN P System con siete neuronas, con las conexiones unidireccionales que indican las fechas y que presentan los spikes que indica el número de la parte derecha del texto que figura en el centro de cada neurona.

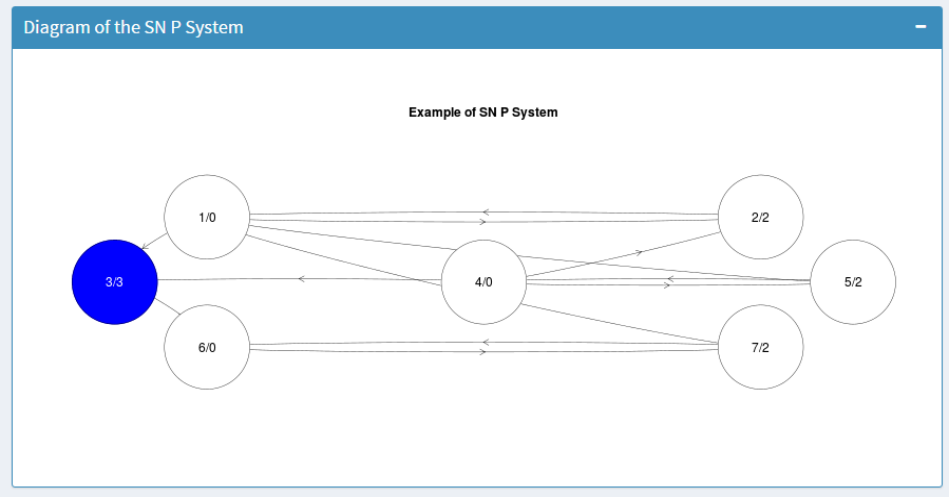


Figura 22 Diagrama que muestra correctamente el SN P System cargado

7.3 Ejecución de un ciclo de la computación de un SN P System

Tras realizar la ejecución de un ciclo, en la Figura 23 comprobamos que la tabla de movimientos refleja los movimientos de spikes del sistema debido a la ejecución de las reglas posibles.

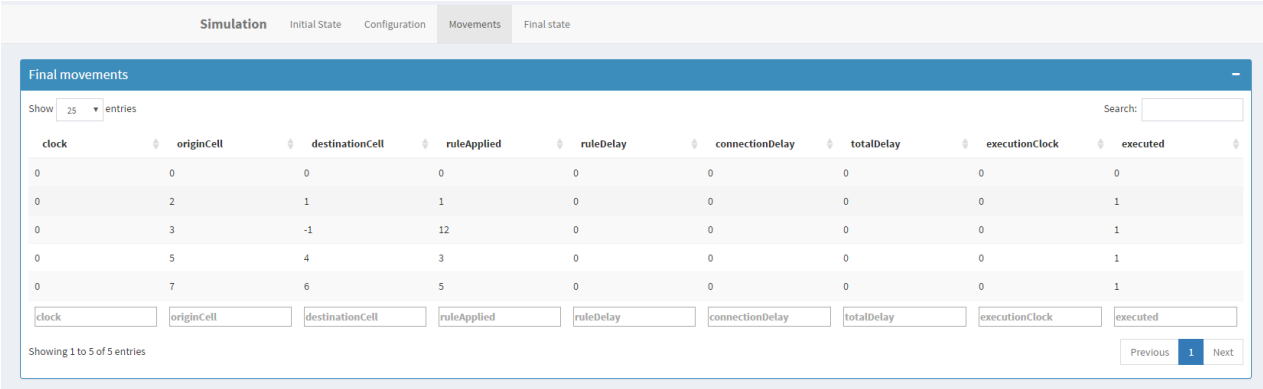


Figura 23 Tabla que muestra los movimientos ejecutados en un ciclo

A continuación, se probarán los diferentes ejemplos de SN P Systems estudiados a lo largo de este trabajo y mostraremos el estado final de sus neuronas para comprobar el correcto funcionamiento del simulador.

7.4 Simulación de un SN P System con siete neuronas

Tomando como ejemplo el SN P System que se probaba en la carga de ficheros al sistema, en la Figura 24 se puede comprobar el estado inicial de sus neuronas. En la **Figura 25** se muestra el estado en el primer ciclo, en la Figura 26 el estado en el segundo y en la Figura 27 el estado final.

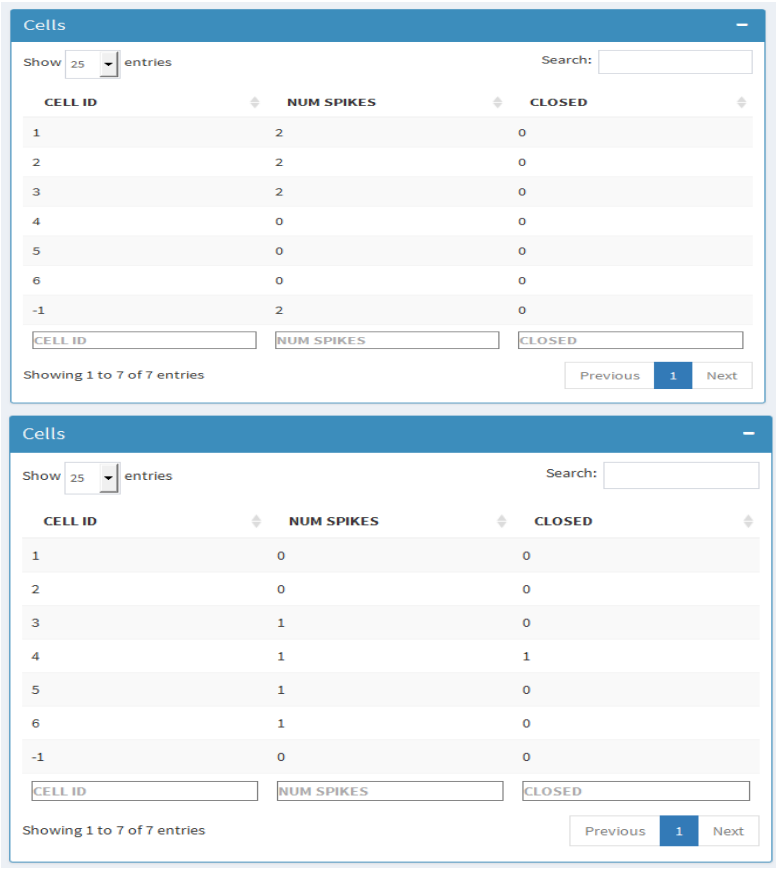


Figura 25 Estado de las neuronas al comienzo del primer ciclo de la simulación del SN P System del apartado 7.4

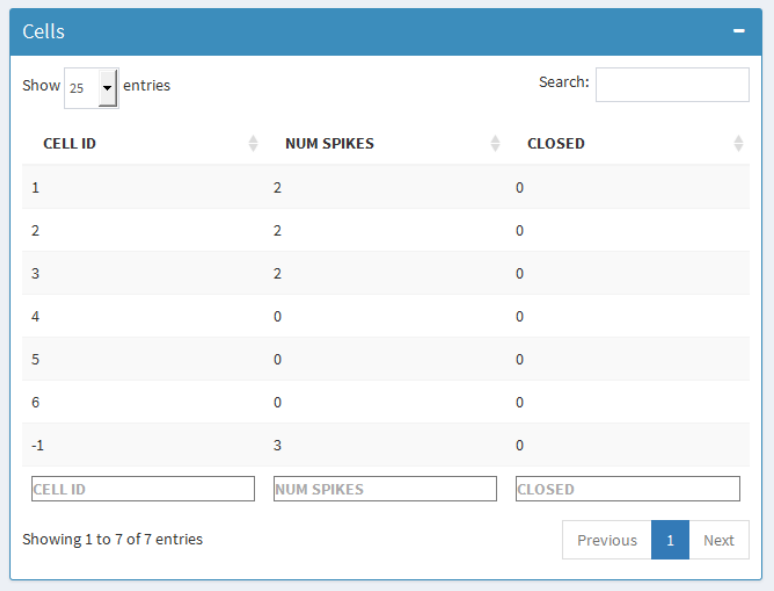


Figura 26 Estado de las neuronas al comienzo del segundo ciclo de la simulación del SN P System del apartado 7.4

Como el sistema tiene una neurona con una regla que presenta retraso en su ejecución, mostramos en la Figura 27 el estado tras su ejecución y los siguientes ciclos en la Figura 28 y en la Figura 29.

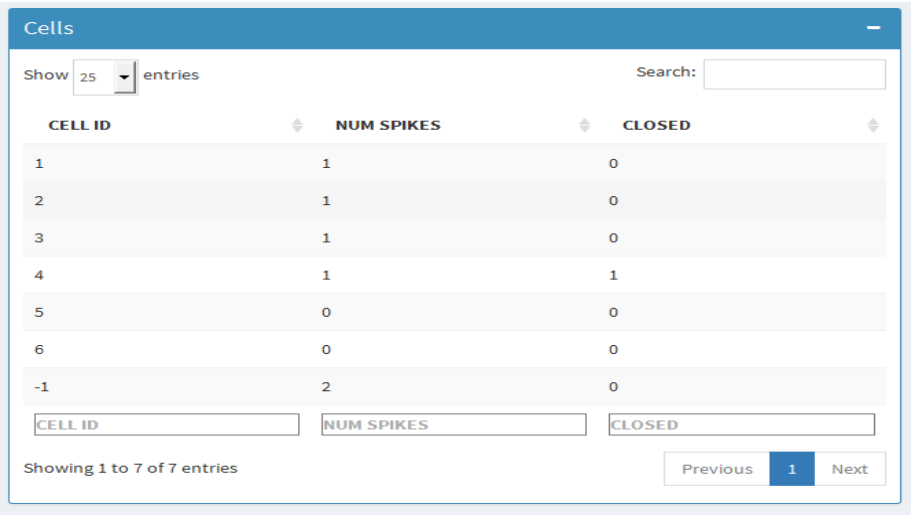


Figura 27 Estado de las neuronas al comienzo del segundo ciclo cuando se ha aplicado una regla con retraso en la simulación del SN P System del apartado 7.4

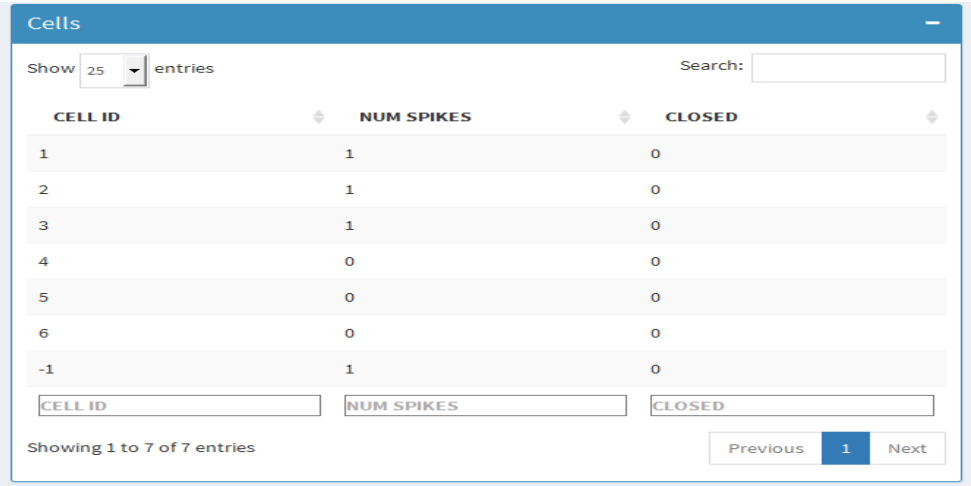


Figura 28 Estado de las neuronas al comienzo del tercer ciclo cuando se ha aplicado una regla con retraso en la simulación del SN P System del apartado 7.4

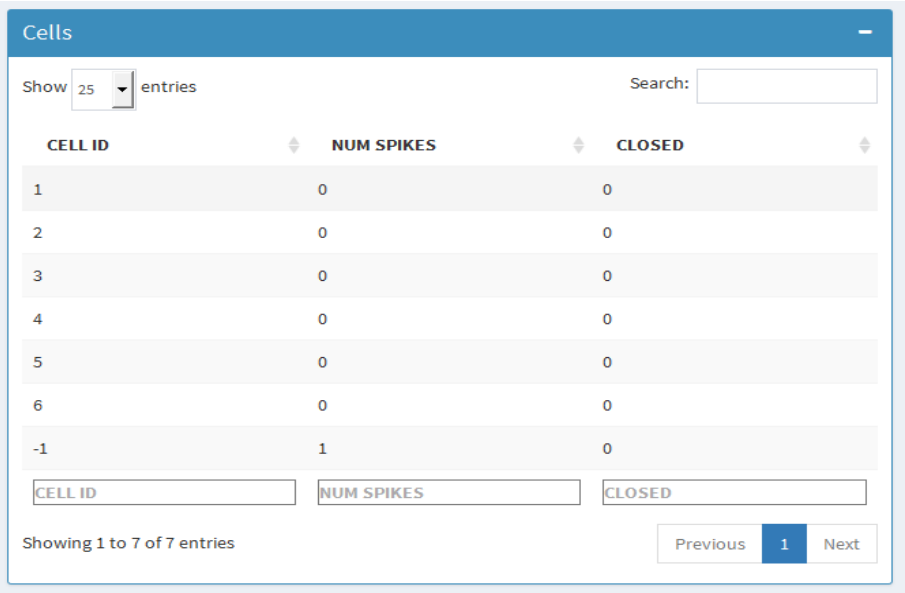


Figura 29 Estado de las neuronas al comienzo del cuarto ciclo cuando se ha aplicado una regla con retraso en la simulación del SN P System del apartado 7.4

7.5 Simulación de puertas y circuitos lógicos

A continuación se mostrarán ejemplos del funcionamiento de los SN P Systems que simulan puertas y circuitos lógicos.

7.5.1 Simulación de las puertas lógicas NOT, AND y OR

Se dará el estado de las neuronas para la entrada de los sistemas y sus correspondientes salidas.

7.5.1.1 Simulación de la puerta lógica NOT

Sabiendo que la entrada es *true*, en la neurona de entrada se añadirá un spike más del que ya existía en ella. El resultado de los spikes en cada neurona al comienzo de la simulación se muestra en la Figura 30.

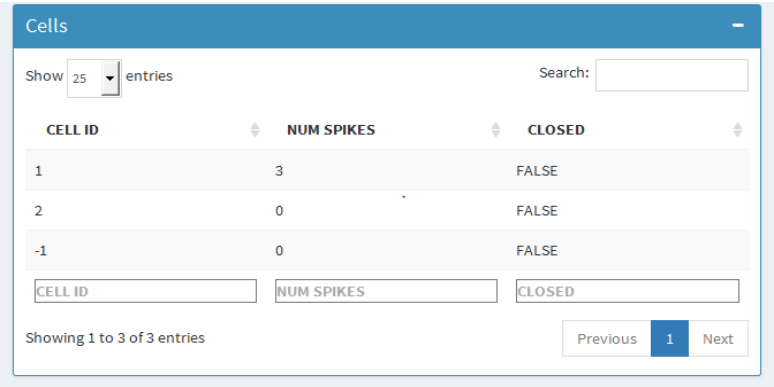


Figura 30 Estado de las neuronas al comienzo de la simulación de la puerta lógica NOT cuya entrada es true

En la Figura 31 se muestra el siguiente ciclo de ejecución y en la Figura 32 se muestra el resultado final. Como la entrada era *true* a este sistema que simula la puerta NOT, la salida correcta que se muestra es *false*.

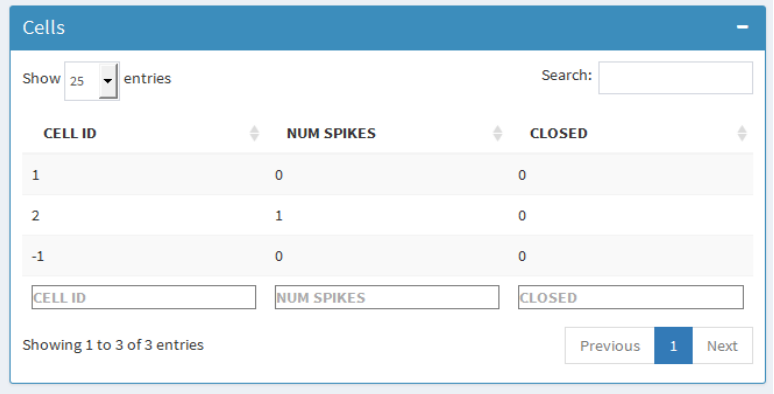


Figura 31 Estado de las neuronas al comienzo del primer ciclo de la simulación de la puerta lógica NOT cuya entrada es true

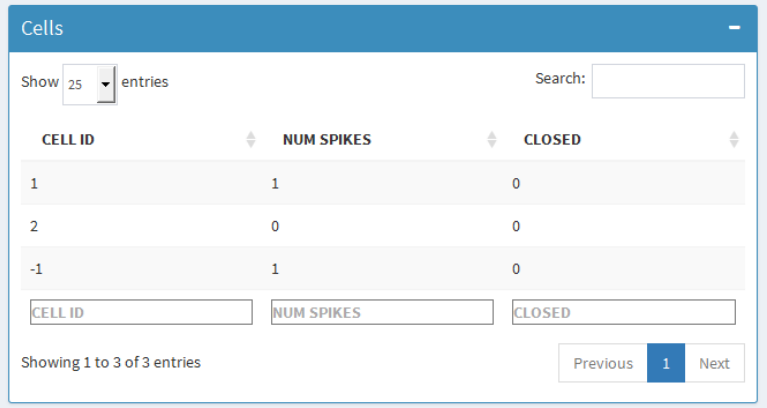


Figura 32 Estado de las neuronas al final de la simulación de la puerta lógica NOT cuya entrada es true

En el caso de que en la entrada de la puerta NOT sea false (Figura 33), se comprueba en la Figura 34 que es correcta la simulación y la salida es *true*.

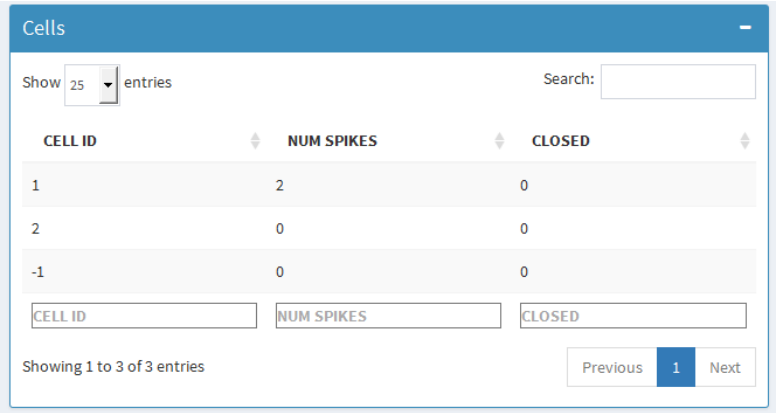


Figura 33 Estado de las neuronas al comienzo de la simulación de la puerta lógica NOT cuya entrada es false

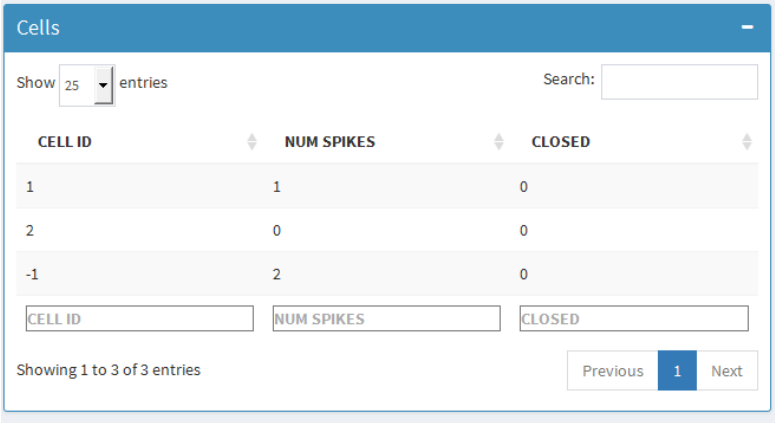


Figura 34 Estado de las neuronas al final de la simulación de la puerta lógica NOT cuya entrada es false

7.5.1.2 Simulación de la puerta lógica AND

Suponiendo que las entradas son *false* y *false*, en la neurona de entrada se añadirán dos (uno y uno) spikes más en ella (Figura 35). El resultado de los spikes en cada neurona al final de la simulación se muestra en la Figura 36.

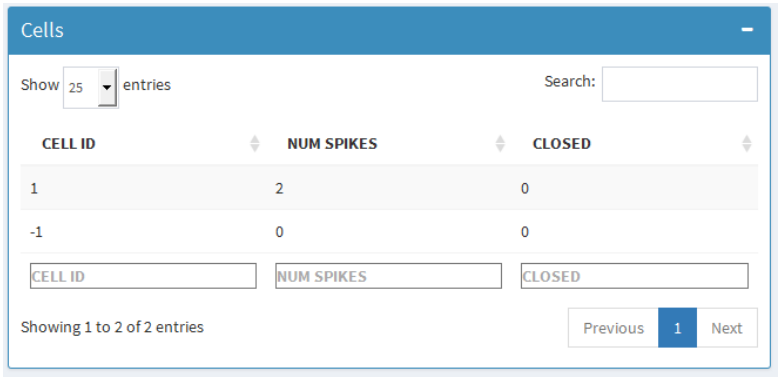


Figura 35 Estado de las neuronas al comienzo del primer ciclo de la simulación de la puerta lógica AND cuyas entradas son false

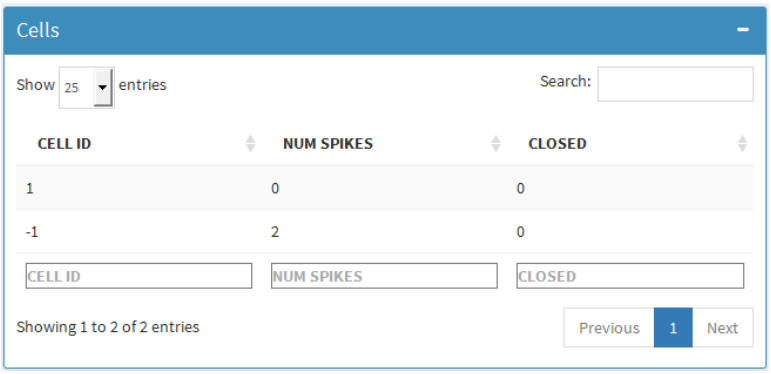


Figura 36 Estado de las neuronas al final de la simulación de la puerta lógica AND cuyas entradas son false

7.5.1.3 Simulación de la puerta lógica OR

Suponiendo que las entradas son *false* y *false*, en la neurona de entrada se añadirán dos (uno y uno) spikes más en ella (Figura 37). El resultado de los spikes en cada neurona al final de la simulación se muestra en la Figura 38.

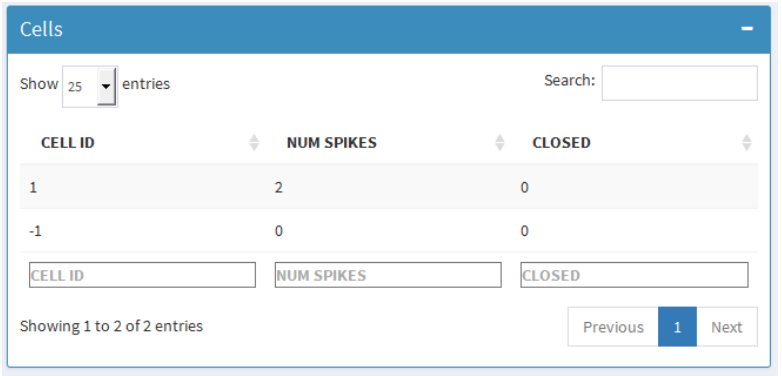


Figura 37 Estado de las neuronas al comienzo del primer ciclo de la simulación de la puerta lógica OR cuyas entradas son false

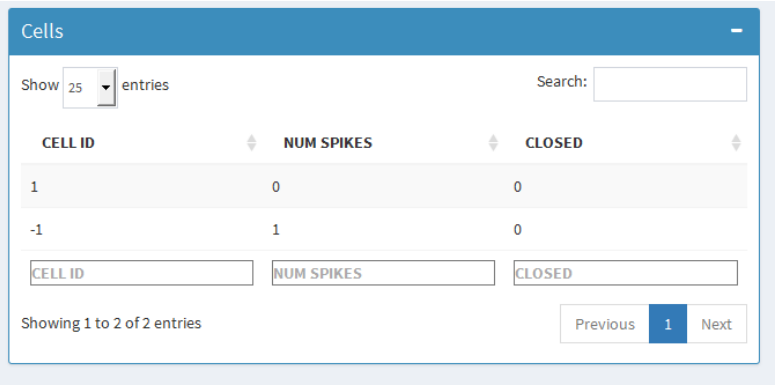


Figura 38 Estado de las neuronas al final de la simulación de la puerta lógica OR cuyas entradas son false

7.5.2 Simulación de un circuito lógico

En la Figura 39 se muestra la entrada al SN P System que va a simular la función lógica tomada de ejemplo en el apartado 4.1.2, cuyas entradas para las 4 variables serán: *false*, *false*, *true* y *false*. En la Figura 40 se muestra el diagrama generado por el sistema para el circuito con dicha entrada y en la Figura 41 se muestra que su salida esperada es la correcta: *false*.

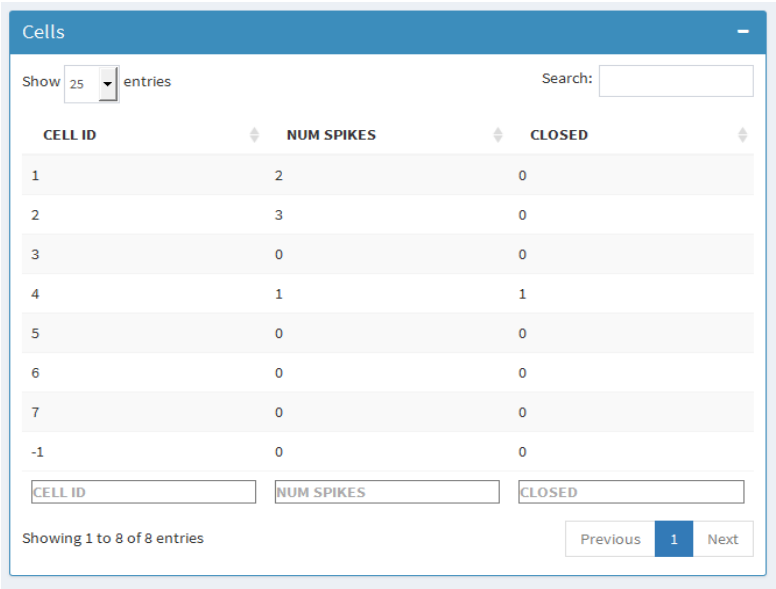


Figura 39 Estado inicial de las neuronas del SN P System que simula el circuito lógico visto en el apartado 4.1.2 y cuyas entradas son *false*, *false*, *true*, y f*alse*

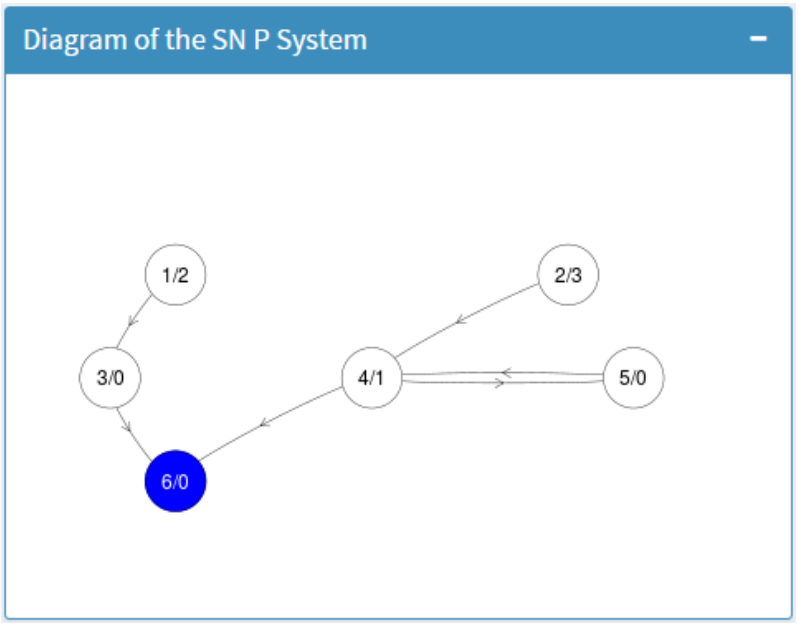


Figura 40 Diagrama del SN P System que simula el circuito lógico visto en el apartado 4.1.2 y cuyas entradas son *false*, *false*, *true*, y *false*

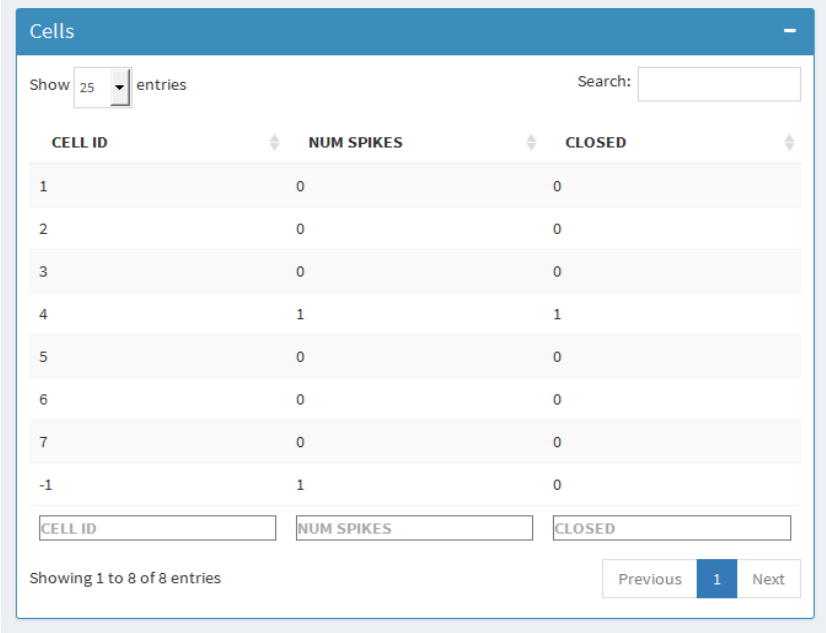


Figura 41 Estado final de las neuronas del SN P System que simula el circuito lógico visto en el apartado 4.1.2 y cuyas entradas son *false*, *false*, *true*, y f*alse*

En la Figura 42 se muestra la entrada al SN P System que va a simular la función lógica tomada de ejemplo en el apartado 4.1.2, cuyas entradas para las 4 variables serán: *true*, *true*, *true* y *false*. En la Figura 43 se muestra que su salida esperada es la correcta: *true*.

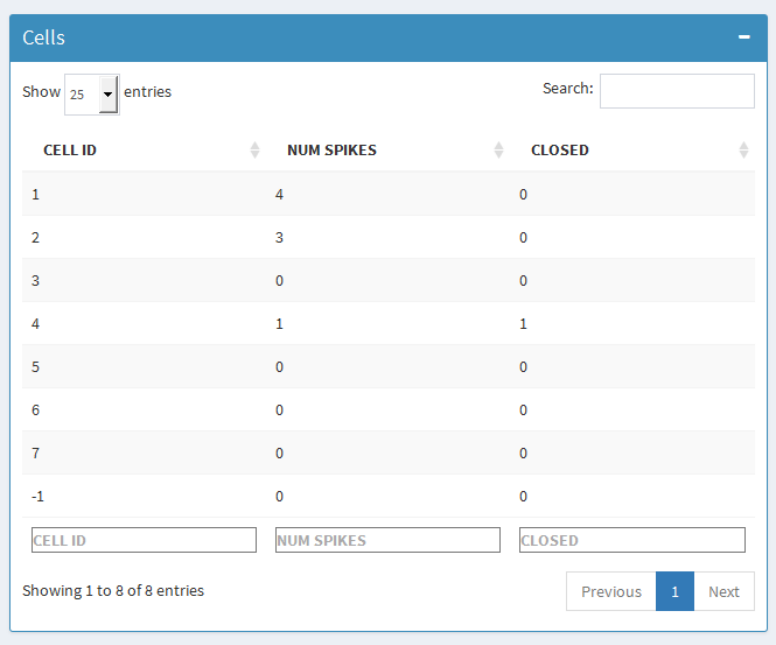


Figura 42 Estado inicial de las neuronas del SN P System que simula el circuito lógico visto en el apartado 4.1.2 y cuyas entradas son *true, true, true y false*

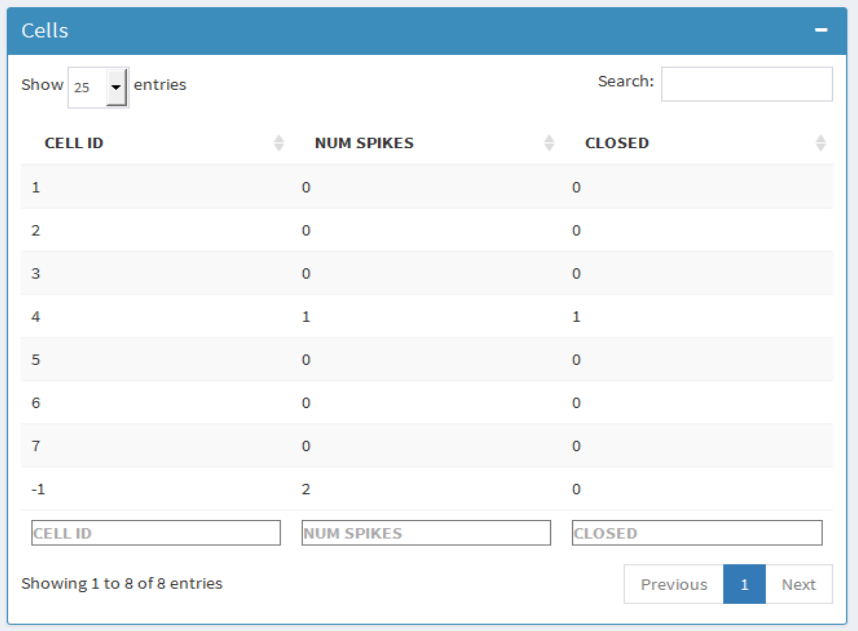


Figura 43 Estado final de las neuronas del SN P System que simula el circuito lógico visto en el apartado 4.1.2 y cuyas entradas son *true, true, true y false*

7.6 Simulación de un SN P System que ordena números enteros

A continuación se mostrarán ejemplos del funcionamiento de un SN P System que ordena números enteros.

7.6.1 Simulación de un SN P System que ordena cinco enteros

En la Figura 44 se muestra el estado inicial del sistema que genera el SN P System. En la Figura 45 se muestra el estado inicial de las neuronas de entrada de un sistema que ordena cinco números: 5, 4, 2, 3 y 1. Por último, en la Figura 46 se muestra el estado final esperado.

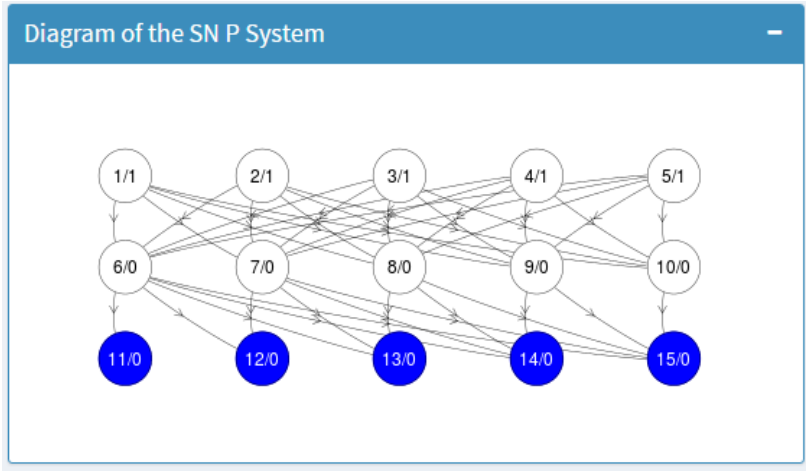
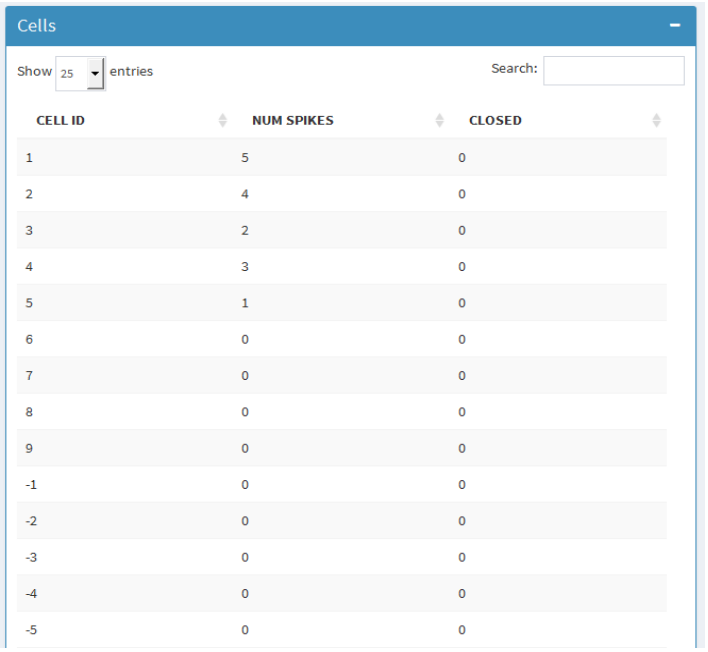


Figura 44 Diagrama del SN P System que ordena los números enteros 5, 4, 2, 3 y 1.



CELL ID	NUM SPIKES	CLOSED
1	5	0
2	4	0
3	2	0
4	3	0
5	1	0
6	0	0
7	0	0
8	0	0
9	0	0
-1	0	0
-2	0	0
-3	0	0
-4	0	0
-5	0	0

Figura 45 Estado inicial de las neuronas del SN P System que ordena los números enteros 5, 4, 2, 3 y 1.

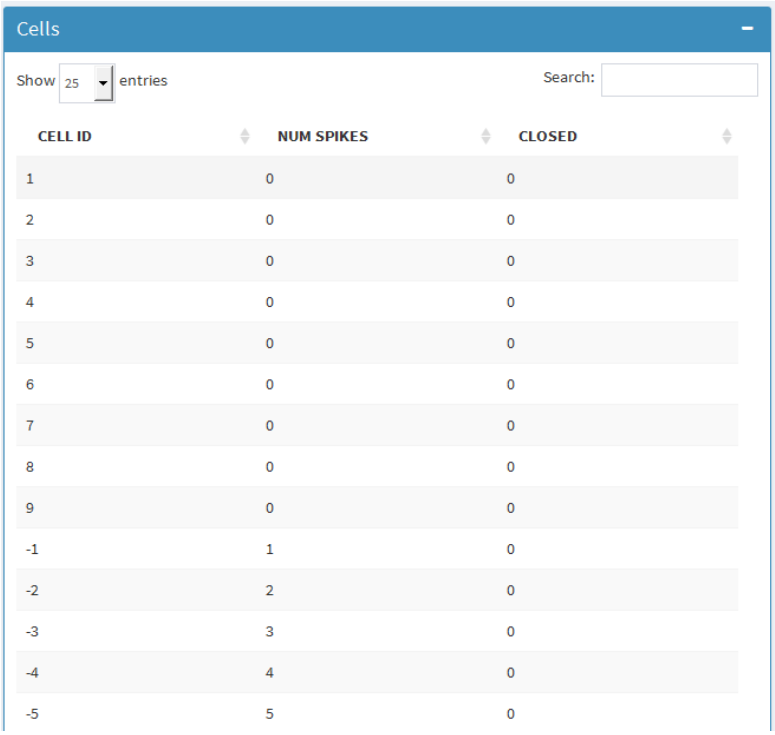


Figura 46 Estado final de las neuronas del SN P System que ordena los números enteros 5, 4, 2, 3 y 1.

7.6.2 Simulación de un SN P System que ordena diez enteros

En la Figura 47 se muestra el estado inicial de las neuronas de entrada de un sistema que ordena cinco números: 5, 4, 2, 3, 1, 7, 6, 9 10 y 8. En la Figura 48 se muestra el estado final esperado.

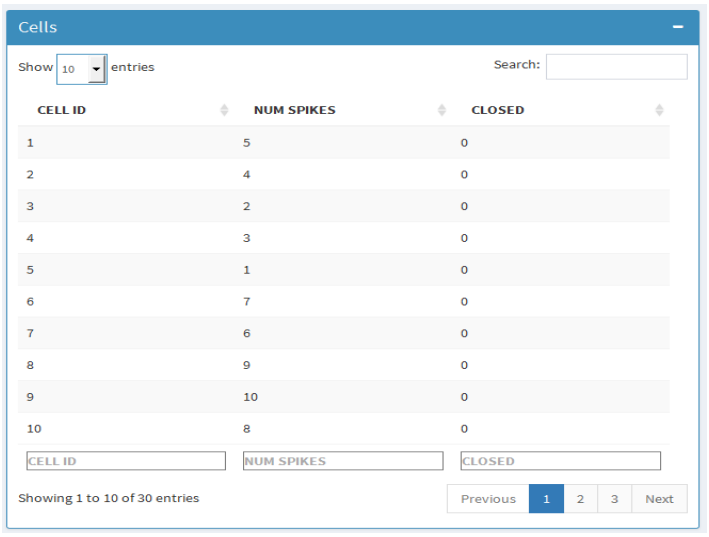


Figura 47 Estado inicial de 10 de las 30 neuronas del SN P System que ordena los números enteros 5, 4, 2, 3, 1, 7, 6, 9, 10 y 8

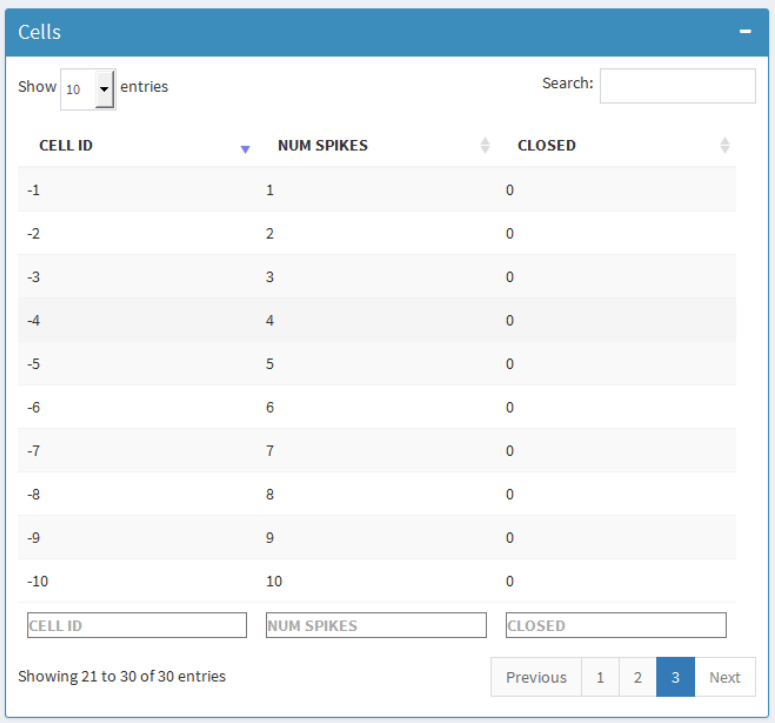


Figura 48 Estado final de las 10 neuronas de salida del SN P System que ordena los números enteros 5, 4, 2, 3, 1, 7, 6, 9, 10 y 8

8 Conclusiones y trabajo futuro

8.1 Conclusiones

Los sistemas de membranas y, más concretamente, los SN P Systems, son dispositivos de cómputo recientemente propuestos en el campo de la Computación Natural. La investigación actual en el área es eminentemente teórica, pero los resultados parecen esperanzadores, por lo que se considera importante disponer de herramientas automatizadas o simuladores que permitan trabajar con estos dispositivos. Con esta idea surge el presente trabajo, en el marco de la realización de un simulador de SN P Systems que sea sencillo y flexible de utilizar. Una vez realizado el sistema es hora de revisar los objetivos que se marcaron al comienzo y analizar su cumplimiento, tarea a la que se dedican los párrafos siguientes.

Respecto a la implementación de un simulador capaz de realizar el diseño de un SN P System o recibirlo como entrada, para procesarlo y mostrar su salida en función de los parámetros dados, se ha conseguido llevar a cabo una aplicación que lo permite.

En cuanto al objetivo de que la simulación de estos sistemas, bien de forma automática o bien permitiendo que el usuario elija las reglas concretas para la ejecución, habilite al usuario para probar resultados teóricos establecido hasta la fecha o elaborar unos nuevos, también se considera logrado, ya que la simulación hace posible analizar y estudiar estas propuestas, al menos en el campo de dos aplicaciones de particular interés, como son los circuitos y puertas lógicas y la ordenación de números naturales. Se entiende que la consecución de este tipo de aplicaciones puede extenderse a otros resultados, tanto existentes como futuros.

Respecto a la forma de mostrar la salida de la simulación, se ha realizado de forma gráfica y a través de tablas que muestran toda la información del sistema. Sin embargo, la edición dentro de la herramienta no es posible, por lo que los cambios han de realizarse en los ficheros de entrada para después cargarlos de nuevo en el sistema. No se considera esta restricción importante, ya que la edición de los ficheros es muy sencilla con un simple procesador de texto.

Por todo lo anterior, se considera que con la realización de este simulador se permite estudiar de forma práctica, sencilla y real, tanto los resultados actuales como los futuros, en el campo de los SN P Systems.

8.2 Trabajo futuro

En base al diseño de este simulador de SN P Systems quedan abiertas muchas funcionalidades que pueden llevarse a cabo para la computación de uno de estos sistemas. Los trabajos futuros que se consideran interesantes son los siguientes:

- Permitir que el simulador trabaje con distintos tipos de spikes, es decir, con un alfabeto que contenga más de un elemento.

- Permitir que existan neuronas que se hallen dentro de otras, dando lugar a diferentes niveles en el sistema.
- Permitir la construcción completa del SN P System desde el simulador, sin tener que cargar un fichero.
- Permitir que la construcción y edición del SN P System se realice a través de un módulo gráfico que permita ir creando el sistema de forma interactiva.
- Realizar un aumento del tipo de reglas posibles que puede albergar el sistema.

Referencias

[1] P Systems, Wikipedia, https://en.wikipedia.org/wiki/P_system.

[2] M. Ionescu, «Some Applications of Spiking Neural P Systems,» Romania, 2007.

[3] D. Díaz-Pernial, F. Peña-Cantillana y M. A. Gutiérrez-Naranjo, «Skeletonizing Images by Using Spiking Neural P Systems,» Sevilla.

[4] M. Ionescu, G. Paun y T. Yokomori, «Spiking Neural P Systems».

[5] Conjuntos semilineales, Wikipedia, https://en.wikipedia.org/wiki/Generalized_arithmetic_progression#semilinear_set.

[6] *P-Lingua,* Wikipedia, www.wikipedia.org/wiki/P_system.

[7] Paralel simulators for Membrane Computing on the GPU, 2015, https://sourceforge.net/projects/pmcgpu/.

[8] D. Ramírez-Martínez y M. A. Gutiérrez-Naranjo, «A Software Tool for Dealing with Spiking Neural P Systems,» Universidad de Sevilla.

[9] MeCoSim, Membrane Computing Simulator, http://www.p-lingua.org/mecosim/.

[10] MP Virtual Laboratory, metaPlab: 2007, http://mplab.sci.univr.it/.

[11] R Studio, https://www.rstudio.com/, IDE.

Glosario

P Systems: los P Systems se definen como una serie de membranas que contienen químicos (en cantidades finitas), catalizadores y reglas que determinan los posibles caminos que pueden tomar los químicos cuando reaccionan. Las reglas también pueden causar que los químicos salgan al exterior de la membrana o que ésta se disuelva

SN P System: son un tipo de P System donde se dan las siguientes características. Las membranas que forman el sistema se encontrarán todas en el entorno, en un mismo nivel, es decir, no se permite que existan células dentro de otras células. En adelante, debido a este hecho, podremos referiremos a tales membranas como neuronas. La segunda característica es que el alfabeto que se va a emplear lo formará un único elemento que representaremos como "a".

Computación con membranas: es un paradigma de computación inspirado en la estructura y el funcionamiento de las células de los organismos vivos. Su objetivo es observar la célula y diseñar máquinas o dispositivos que proporcionan herramientas novedosas para atacar, principalmente, problemas relevantes de la vida real, como construir modelos computacionales de procesos biológicos.

NRE: Definiremos a la familia de los lenguajes recursivamente numerables como RE y a la familia de todos los conjuntos de números naturales Turing – computables como NRE.

Anexos

A. Requisitos funcionales. Especificaciones de casos de uso.

Identificador: CU01	**Nombre Caso de Uso:** Carga de un SN P System
Actores involucrados: Usuario	
Resumen: Permite cargar el diseño de un SN P System facilitando tres ficheros: uno fichero con los spikes, otro fichero con las reglas que contiene cada neurona y otro fichero con las conexiones entre las diferentes neuronas del sistema.	
Pre-condiciones: El usuario se encuentra en la ventana principal de la aplicación, concretamente en la zona de carga de ficheros.	
Post-condiciones: Se habilita al usuario el apartado de simulación en el sistema.	
Curso Básico de Eventos	
Usuario	**Sistema**
1.- El usuario accede a la pantalla principal, donde figura el área de carga de la información.	2.- El sistema muestra los diferentes directorios para cada uno de los archivos a cargar.
3.- El usuario selecciona el directorio y los ficheros que ha de cargar en el sistema: spikes.csv, rules.csv y connections.csv (estos son los nombres	4.- El sistema solicita al usuario seleccionar las características de cada uno de los archivos cargados: cabecera, separador, quote.

de los ficheros por defecto)	
5.- El usuario indica las características de los archivos: cabecera, separador y quote.	6.- El sistema carga los ficheros y habilita el apartado de simulación.
Caminos Alternativos	
Caminos de Excepción	
Si el usuario ha introducido algún fichero con una información incorrecta o que no cumple las características requeridas, el sistema informa al usuario y pasa al estado inicial del caso de uso, descrito por la pre-condición.	

Identificador: CU02	**Nombre Caso de Uso:** Mostrar el estado inicial del SN P System
Actores involucrados: Usuario	
Resumen: Se muestra a través de un diagrama y tablas la información del SN P System cargado previamente de acuerdo al CU01.	
Pre-condiciones: El usuario ha realizado la carga de la información del sistema y el sistema ha habilitado el módulo de simulación de la aplicación.	
Post-condiciones: El usuario puede volver a cargar otra información o proceder a establecer la configuración de la simulación.	

Curso Básico de Eventos	
Usuario	**Sistema**
1.- El usuario accede al módulo de simulación.	2.- El sistema muestra el diagrama del SN P System cargado previamente y las reglas que contiene cada neurona.
Caminos Alternativos	
Caminos de Excepción	

Identificador: CU03	**Nombre Caso de Uso:** Configurar la simulación del SN P System
Actores involucrados: Usuario	
Resumen: Configura los diferentes parámetros para llevar a cabo la simulación del SN P System cargado: tipo de sistema, tipo de salida del sistema y si se desea exportar la información obtenida.	
Pre-condiciones: El usuario ha realizado la carga de la información del sistema, el sistema ha habilitado el módulo de simulación de la aplicación, el usuario ha accedido, dentro del área de simulación, al apartado de configuración.	
Post-condiciones:	

Se permita al usuario simular el SN P System cargado con los parámetros de configuración indicados.	
Curso Básico de Eventos	
Usuario	**Sistema**
1.- El actor accede al área de configuración de la simulación.	2.- El sistema muestra los diferentes parámetros a determinar: tipo de sistema (normal, total spiking o decaying spikes), tipo de salida (spikes en la neurona de salida o tiempo transcurrido entre dos emisiones de esta neurona), si desea exportar la información o no.
3.- El actor introduce los diferentes parámetros a determinar: tipo de sistema (normal, total spiking o decaying spikes), tipo de salida (spikes en la neurona de salida o tiempo transcurrido entre dos emisiones de esta neurona), si desea exportar la información o no.	4.- El sistema guarda los parámetros introducidos.
Caminos Alternativos	
Caminos de Excepción	
El sistema muestra un error de carga y vuelve al estado inicial si se introduce un fichero erróneo en cualquiera de los tres apartados: spikes, reglas o conexiones.	

Identificador: CU04	**Nombre Caso de Uso:** Simulación ciclo a ciclo del SN P System
Actores involucrados:	

<table>
<tr><td colspan="2">Usuario</td></tr>
<tr><td colspan="2">Resumen:

Realiza la operativa de ejecución de un ciclo de reloj para el sistema precargado anteriormente y permite continuar realizando esta tarea hasta que el usuario desee.</td></tr>
<tr><td colspan="2">Pre-condiciones:

El usuario ha cargado la información del sistema, ha elegido los diferentes parámetros de la configuración y se encuentra en el área de simulación por pasos.</td></tr>
<tr><td colspan="2">Post-condiciones:

Permite al usuario realizar de nuevo la simulación para otro ciclo de simulación.</td></tr>
<tr><td colspan="2">Curso Básico de Eventos</td></tr>
<tr><td>Usuario</td><td>Sistema</td></tr>
<tr><td>1.- El usuario accede al área de simulación por pasos de la aplicación.</td><td>2.- El sistema muestra el ciclo actual en que se encuentra la simulación e indica al usuario que introduzca el siguiente ciclo a ejecutar y, si es necesario elegir una regla a usar porque hay varias posibles, indica al usuario que especifique la selección correspondiente.</td></tr>
<tr><td>3.- El actor introduce el nuevo ciclo a simular y las reglas a usar, en caso de que sea necesario.</td><td>4.- El sistema calcula el nuevo estado tras la simulación de dicho ciclo y muestra el diagrama actual y las tablas con su información.</td></tr>
<tr><td colspan="2">Caminos Alternativos</td></tr>
<tr><td colspan="2"></td></tr>
<tr><td colspan="2">Caminos de Excepción</td></tr>
<tr><td colspan="2"></td></tr>
</table>

Identificador: CU05	**Nombre Caso de Uso:** Simular número concreto de ciclos del SN P System
Actores involucrados: Usuario	
Resumen: Realiza la operativa de ejecución del número de ciclos de reloj que el usuario indique para el sistema precargado anteriormente.	
Pre-condiciones: El usuario ha cargado la información del sistema, ha elegido los diferentes parámetros de la configuración y se encuentra en el área de simulación de un número concreto de casos.	
Post-condiciones: Permite al usuario realizar de nuevo la simulación para otro número de ciclos de simulación o elegir la simulación por pasos, o cargar otro sistema.	
Curso Básico de Eventos	
Usuario	**Sistema**
1.- El usuarioaccede al área de simulación de un número concreto de pasos de la aplicación.	2.- El sistema muestra el ciclo actual en que se encuentra el sistema e indica al usuario que introduzca el ciclo final que desea ejecutar.
3.- El actor introduce el número de ciclos a simular.	4.- El sistema calcula el nuevo estado tras la simulación de dicho número de ciclos y muestra el diagrama actual y las tablas con su información.
Caminos Alternativos	

Caminos de Excepción

Identificador:	**Nombre Caso de Uso:**
CU6	Mostrar movimientos simulación SN P System

Actores involucrados:

Usuario

Resumen:

Permite mostrar los movimientos que la aplicación ha realizado para una simulación solicitada por el usuario.

Pre-condiciones:

El usuario ha realizado al menos un tipo de simulación, por lo que el sistema ya ha ejecutado un ciclo y cuenta con la aplicación abierta en el área de simulación o cargar otro sistema.

Post-condiciones:

Se permite al usuario realizar de nuevo la simulación para otro número de ciclos de simulación o elegir/continuar la simulación por pasos o cargar otro sistema.

Curso Básico de Eventos

Usuario	**Sistema**
1.- El actor accede al área de movimientos.	2.- El sistema muestra los movimientos realizados hasta el momento de la ejecución y que han sido almacenados en la aplicación.

Caminos Alternativos
Caminos de Excepción

Identificador: CU7	**Nombre Caso de Uso:** Exportar el resultado de la simulación
Actores involucrados: Usuario	
Resumen: Permite guardar la información generada por la simulación con el fin de poder ser utilizada como entrada del sistema.	
Pre-condiciones: El usuario ha elegido en la configuración la opción de exportación de la información y ha realizado al menos un tipo de simulación, por lo que el sistema ya ha ejecutado un ciclo.	
Post-condiciones: Se permite al usuario realizar de nuevo la simulación para otro número de ciclos de simulación o elegir/continuar la simulación por pasos o cargar otro sistema.	
Curso Básico de Eventos	
Usuario	**Sistema**
1.- El usuarioaccede al área de	2.- El sistema muestra los diferentes

configuración de la simulación.	parámetros a determinar: tipo de sistema (normal, total spiking o decaying spikes), tipo de salida (spikes en la neurona de salida o tiempo transcurrido entre dos emisiones de esta neurona), y si desea o no exportar la información.
3.- El actor introduce los diferentes parámetros a determinar: tipo de sistema (normal, total spiking o decaying spikes), tipo de salida (spikes en la neurona de salida o tiempo transcurrido entre dos emisiones de esta neurona), e indica que sí desea exportar la información.	4.- El sistema guarda los parámetros introducidos y cuando el usuario realice cualquier simulación, guardará el estado de cada ciclo de ejecución.
Caminos Alternativos	
Caminos de Excepción	